IMPELEMENTASI *HARDWARE REDUNDANCY* PADA *SWITCHING* PINTU OTOMATIS DENGAN METODE *COLD STANDBY* DAN *WATCHDOG*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
Muhammad Rifqi Muzaki
NIM : 145150300111009

PROGRAM STUDI TEKNIK KOMPUTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPELEMENTASI *HARDWARE REDUNDANCY* PADA *SWITCHING* PINTU OTOMATIS DENGAN METODE *COLD STANDBY* DAN *WATCHDOG*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :
Muhammad Rifqi Muzaki
NIM: 145150300111009

Skripsi ini telah diuji dan dinyatakan lulus pada
03 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sabriansyah Rizqika Akbar, S.T, M.Eng	Widhi Yahya, S.Kom., M.Sc.
NIP: 19820809 201212 1 004	NIK: 2016078911211001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 18 Juli 2018

Muhammad Rifqi Muzaki
NIM: 145150300111009

KATA PENGANTAR

Dengan menyebut nama Allah SWT yang Maha pengasih dan Maha penyayang. Penulis memanjatkan puja dan puji syukur atas segala limpahan rahmat yang diberikan sehingga laporan skripsi yang berjudul "IMPELEMENTASI *HARDWARE REDUNDANCY* PADA *SWITCHING* PINTU OTOMATIS DENGAN METODE *COLD STANDBY* DAN *WATCHDOG*" ini dapat terselesaikan. Karya ilmiah ini dapat terselesaikan karena dukungan dan bantuan dari berbagai pihak yang terlibat. Untuk itu penulis ingin menyampaikan rasa hormat dan terimakasih kepada:

1. Ibu Luluk Ismunah dan bapak Ahmad Layin selaku kedua orangtua penulis yang telah melahirkan dan membesarkan penulis, serta terus memberikan doa dan dukungan untuk penyelesaian naskah skripsi ini.
2. Bapak Wayan Firdaus Mahmudy, S.Si., M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika.
4. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng selaku Ketua Program Studi Teknik Informatika sekaligus dosen pembimbing I yang telah memberikan bimbingan selama penyusunan skripsi ini.
5. Bapak Widhi Yahya, S.Kom., M.Sc. selaku dosen pembimbing II yang selalu memberikan arahan kepada penulis sehingga dapat menyelesaikan skripsi ini.
6. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan kepada penulis selama menempuh studi di Universitas Brawijaya selama ini.
7. Teman – teman penulis Ary, Adib, Didik, Lamidi, Ayu, Khurin, Imam, Deddy, Irvan, Anggi, dan keluarga besar Teknik Komputer 2014 yang telah memberi dukungan dan semangat selama pengerjaan skripsi ini.

Penulis menyadari dalam penyusunan skripsi ini masih memiliki banyak kekurangan. Oleh karena itu penulis membutuhkan kritik dan saran dalam skripsi ini. Akhir kata penulis berharap semoga kedepannya skripsi ini bermanfaat.

Malang, 18 Juli 2018
Penulis

Muhammad Rifqi Muzaki
mrifqi1101@gmail.com

ABSTRAK

Perangkat elektronik pasti memiliki batas masa pakai dan sewaktu – waktu dapat mengalami kerusakan. Sebuah sistem yang terdiri dari beberapa perangkat elektronik akan mengalami kegagalan jika komponen yang krusial mengalami kerusakan. Oleh karena itu komponen krusial dalam sebuah sistem harus lebih handal. Untuk meningkatkan kehandalan sistem sebuah mekanisme seperti *Hardware Redundancy* bisa diterapkan. *Hardware Redundancy* merupakan sebuah mekanisme meningkatkan kehandalan dengan cara menduplikat komponen penting sebagai cadangan. Terdapat beberapa jenis dari mekanisme Hardware Redundancy yang salah satunya adalah *Cold Standby Redundancy* yang menggunakan konsep cadangan tanpa catu daya. Konsep *Cold Standby* adalah menambahkan duplikat komponen penting yang standby tanpa catu daya dan akan aktif ketika komponen penting tersebut mengalami kerusakan. Pada penelitian ini *Cold Standby Redundancy* diterapkan pada sebuah sistem kunci pintu otomatis berbasis Arduino Uno dan RFID. Komponen penting pada sistem tersebut adalah Arduino Uno yang bertindak sebagai kontroler dalam sistem. Implementasi *Cold Standby Redundancy* pada sistem ini menggunakan perangkat pihak ketiga (Arduino Nano) sebagai *fault detector* pada Arduino Uno yang utama (*master*). Melalui komunikasi serial I2C *master* mengirimkan *keep-alive status* secara periodik kepada *fault detector* sebagai tanda apakah masih beroperasi atau sedang mengalami kerusakan. Selain itu terdapat perangkat *switch* yang bertugas mengalihkan kontrol sistem ke *master* atau *slave*. *Fault detector* akan mengirimkan *trigger* untuk melakukan peralihan kepada perangkat *switch* ketika mendeteksi kerusakan pada *master*. Berdasarkan hasil pengujian yang telah dilakukan, kehandalan sistem meningkat setelah menerapkan mekanisme *Cold Standby Redundancy*. Nilai kehandalan sistem meningkat sebesar 15,45% untuk satu tahun waktu beroperasi.

Kata kunci: *Fault Tolerant, Reliability, Hardware Redundancy, Cold Standby Redundancy*

ABSTRACT

Electronic devices must have a lifetime limit and may be damaged at any time. A system consisting of several electronic devices will fail if the crucial component is damaged. Therefore, crucial components in a system should be more reliable. To improve system reliability a mechanism such as Hardware Redundancy can be applied. Hardware Redundancy is a mechanism to improve reliability by duplicating important components as backup. There are several types of Hardware Redundancy mechanisms, one of which is Cold Standby Redundancy which uses the unpowered backup concept. The Cold Standby concept is to add a duplicate of an important component that standby without a power supply and will be active when the component is damaged. In this study Cold Standby Redundancy is applied to an automatic door lock system based on Arduino Uno and RFID. An important component of the system is Arduino Uno which acts as a controller in the system. Implementation of Cold Standby Redundancy on this system uses a third party device (Arduino Nano) as the fault detector of main Arduino Uno (master). Through I2C serial communication the master sends keep-alive status periodically to the fault detector as a sign of whether it is still operating or is being damaged. In addition there is a switch device in charge of switching the system control to master or slave. The fault detector sends the trigger to make a switch to the switch device when it detects damage to the master. Based on the results of tests that have been done, system reliability increases after applying the Cold Standby Redundancy mechanism. System reliability value increased by 15.45% for a year operating time.

Keyword : Fault Tolerant, Reliability, Hardware Redundancy, Cold Standby Redundancy

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Sebuah sistem otomatis dalam segi penggunaan memiliki sisi fungsional dan non-fungsional. Sisi fungsional adalah bagaimana sistem bekerja berdasarkan tujuannya. Sedangkan sisi non fungsional adalah bagaimana sistem dapat tetap berjalan dengan adanya faktor – faktor yang dapat mengakibatkan sisem mengalami kegagalan. Oleh karena itu sebuah sistem harus dapat diandalkan (*dependable)*. Terdapat beberapa atribut dari dependability sebuah sistem. Salah satu atribut utamanya adalah *reliability* (kehandalan). *Reliability* atau jika disimbolkan menjadi *R(t)* merupakan probabilitas jalannya sebuah sistem tanpa adanya kegagalan dalam interval waktu tertentu atau [0, *t*], dimana sistem mulai beroperasi pada waktu ke 0 (Dubrova, 2013).

Pada penelitian ini sistem otomatis yang akan diteliti adalah sistem yang umum dan sederhana yaitu sistem pengunci pintu otomatis berbasis RFID (*Radio-Frequency Identification*) dan Arduino. Dimana sistem ini bekerja berdasarkan pembacaan kartu RFID yang didekatkan pada RFID *reader,* data pada kartu akan dicocokkan dengan data yang tersimpan pada memori program Arduino untuk menentukan otorisasi dari pemilik kartu (Nehete, et al., 2016). Perangkat – perangkat dari sistem kunci pintu otomatis tersebut merupakan perangkat elektronik yang pasti akan mengalami kerusakan. Apabila hal ini terjadi maka akan berdampak pada pintu yang tidak bisa dibuka dan tentunya itu merugikan. Oleh karena itu sebuah sistem pengunci pintu otomatis harus *reliable*. Untuk mewujudkannya pada penelitian ini diimplementasikan mekanisme *hardware redundancy* pada sistem pengunci otomatis berbasis RFID dan Arduino.

Hardware Redundancy adalah teknik redundansi yang dilakukan pada level hardware atau komponen fisik (Dubrova, 2013). Salah satu jenis *hardware redundancy* adalah *standby redundancy*, yaitu unit spare bersifat *standby* dan aktif ketika komponen utama mengalami kerusakan. *Standby redundancy* sendiri memiliki dua metode yaitu *cold standby* dan *hot standby*. Pada *cold standby* unit *standby* yang tidak tercatu daya, sedangkan pada *hot standby* unit *standby* tercatu daya.

Cold standby redundancy adalah sebuah mekanisme menambahkan duplikat dari suatu komponen penting dalam keadaan mati sebagai cadangan yang hanya aktif ketika komponen penting tersebut mengalami kegagalan. *Cold standby* memiliki keunggulan dari segi konsumsi energi dan keandalan unit lebih terjaga karena unit *standby* tidak tercatu daya (National Instruments, 2008). Dengan pertimbangan keunggulan tersebut penelitian akan menerapkan metode *cold-standby redundancy* yang diimplementasikan pada sistem kunci pintu otomatis. Dalam penelitian ini perangkat yang akan diredundansi adalah *controller unit* (Arduino Uno) karena merupakan *critical component* dari sistem. Penelitian ini menggunakan pendekatan mekanisme *Keep-alive* sebagai indikator status kerusakan dengan memanfaatkan komunikasi serial I2C yang ada pada Arduino.

Slave controller dipasang dalam keadaan mati (tidak tercatu daya) dan akan dinyalakan oleh perangkat *fault detector* yang secara periodik menerima sinyal *Keep-alive* dari *master controller*. Ketika sinyal *Keep-alive* yang diterima perangkat *fault detector* dari *master controller* berhenti, maka diindikasikan unit *master controller* mengalami kerusakan atau mati. Pada saat itu terjadi *fault detector* akan men-*trigger switch device* untuk melakukan *switching* dan mengaktifkan *slave controller*.

Menerapkan mekanisme *cold standby redundancy* pada perangkat pengunci pintu otomatis ini diharapkan mampu meningkatkan *reliability* sistem. Adanya perangkat cadangan sebagai pengganti perangkat utama diperkirakan dapat mengurangi resiko dari dampak kegagalan sistem. Dengan demikian dapat diwujudkan sistem pengunci pintu yang dapat diandalkan.

1.2 Rumusan masalah

Berdasarkan permasalahan yang ada pada latar belakang, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana perancangan perangkat keras sistem *hardware redundancy* untuk melakukan *fail recovery* dengan menggunakan metode *Cold Standby*?
2. Bagaimana perancangan perangkat lunak sistem *hardware redundancy* untuk melakukan *fail recovery* dengan menggunakan metode *Cold Standby*?
3. Bagaimana analisis perbandingan dari sistem seri (tanpa redundansi) dan parallel (dengan redundansi)?

1.3 Tujuan

Tujuan umum :

1. Untuk mengetahui bagaimana rancangan sistem kunci pintu otomatis berbasis Arduino yang memiliki *reliability* lebih tinggi.
2. Untuk mengetahui bagaimana kinerja sistem kunci pintu otomatis setelah menerapkan *Cold Standby Redundancy*.

Tujuan khusus :

1. Untuk mengetahui perancangan perangkat keras sistem *hardware redundancy*.
2. Untuk mengetahui perancangan perangkat lunak sistem *hardware redundancy*.
3. Untuk mengetahui bagaimana perbandingan *reliability* sistem tanpa redundansi dan dengan redundansi.

1.4 Manfaat

Manfaat yang diharapkan dari adanya karya tulis ini adalah dapat berkontribusi terhadap dunia teknologi terutama dalam hal menigkatkan kehandalan dari sebuah sistem otomatis. Karya tulis ini diharapkan bisa menjadi referensi kepustakaan institusi pendidikan terutama di Universitas Brawijaya. Karya tulis ini dapat memberikan wawasan bagi mahasiswa – mahasiswa lain yang mungkin nantinya akan melakukan penelitian di bidang yang sama.

1.5 Batasan masalah

Batasan – batasan masalah diperlukan untuk menjaga ruang lingkup penelitian agar lebih terarah dan hasil penelitian yang lebih jelas. Berikut beberapa batasan masalah dalam penelitian ini :

1. Penelitian ini dibatasi pada lingkup analisis *reliability* sistem.
2. Penelitian ini hanya difokuskan pada redundansi terhadap salah satu komponen dari sistem dengan satu cadangan.
3. Sistem kunci pintu otomatis yang sudah ada tidak dibahas secara detil, namun hanya berfokus pada bagaimana sistem tersebut bisa lebih *reliable*.
4. Metode yang diterapkan untuk membuat sistem lebih reliable adalah metode *Cold Standby Redundancy*.
5. Analisis menggunakan teknik evaluasi dan pemodelan sistem dengan *Reliability* Block Diagram.

1.6 Sistematika pembahasan

Sistematika penulisan pada skripsi ini terdiri dari beberapa bab sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan skripsi.

BAB 2 KAJIAN PUSTAKA

Bab ini berisi teori – teori pendukung tentang rumah pintar, fault tolerance, redundansi, dan arduino.

BAB 3 METODOLOGI

Bab ini berisi metode penelitian, kebutuhan yang dibutuhkan sistem, metode perancangan dan implementasi, metode pengujian serta pengambilan kesimpulan.

BAB 4 ANALISIS KEBUTUHAN

Bab ini menjelaskan detil analisis kebutuhan yang diperlukan untuk perancangan, implementasi, serta pengujian sistem.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan bagaimana implementasi dari perancangan sistem redundansi terhadap sistem kunci pintu otomatis.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini berisi pelaksanaan pengujian dan analisis hasil pengujian dengan memodelkan sistem menggunakan teknik evaluasi.

BAB 7 PENUTUP

Bab ini berisi kesimpulan yang diambil dari hasil penelitian beserta saran untuk keperluan penelitian lanjutan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini terdiri dari beberapa kebutuhan kepustakaan yang mendukung untuk pelaksanaan penelitian. Beberapa bahan pustaka akan ditinjau dalam tinjauan pustaka yang membahas beberapa penelitian yang sudah ada dan relevan dengan penelitian yang akan dilakukan. Dasar – dasar teori yang sudah dikumpulkan oleh penulis juga akan dijelaskan pada bab ini.

2.1 Tinjauan Pustaka

Beberapa referensi penelitian sebelumnya yang telah dikumpulkan untuk mendukung penelitian akan dibahas pada sub bab ini. Beberapa penelitian sebelumnya yang relevan dengan penelitian ini dijelaskan perbedaannya dengan penelitian ini. Penelitian – penelitian lampau tersebut diperlukan sebagai tinjauan pustaka untuk mengetahui apa yang dikembangkan dalam penelitian ini.

Pada penelitian "Desain dan Prototipe Kunci Pintu Otomatis Menggunakan RFID Berbasis Arduino Uno" yang dilakukan oleh Roosano dan Purnomo, sistem menggunakan kartu RFID, RFID *reader*, *board* mikrokontroller Arduino Uno serta servo sebagai *actuator* pengunci. Kartu RFID berfungsi sebagai objek pengenal yang di dalamnya terdapat data berupa *ID number*. RFID *reader* digunakan sebagai pembaca informasi pada tag RFID. Arduino Uno sebagai database dan memproses data yang diperoleh dari RFID *reader*. Servo sebagai *actuator* pengunci akan terbuka jika kartu RFID yang dibaca oleh RFID *reader* sesuai dengan database (Roosano & Purnomo, 2016).

Pada penelitian "Rancang Bangun Pengaman Pintu Otomatis Menggunakan *E-KTP* Berbasis Mikrokontroler ATmega 328" yang dilakukan oleh Eko Saputro, sistem menggunakan *E-KTP* sebagai RFID *tag,* RFID *reader,* mikrokontroler ATmega 328, serta solenoid sebagai *actuator* pengunci. RFID *reader* yang digunakan untuk membaca ID *E-KTP* memiliki frekuensi 13,56 MHz dengan jarak pembacaan maksimal 1,8 cm. Solenoid akan terbuka jika ID *E-KTP* sesuai dengan ID yang tersimpan pada memori mikrokontroler ATmega 328 (Saputro & Wibawanto, 2016).

Berdasarkan referensi – referensi tersebut dapat diketahui bahwa terdapat beberapa sistem dalam bidang keamanan seperti kunci pintu otomatis tidak memiliki sifat *fault tolerant*. Pada kedua penelitian tersebut tidak terdapat pengujian tentang bagaimana ketika sistem mengalami kegagalan. Oleh karenanya penulis tertarik untuk mengembangkan sistem kunci pintu menjadi sistem yang *fault tolerant*. Dengan menerapkan metode *Cold Standby Redundancy* pada sistem kunci otomatis berbasis Arduino maka akan memungkinkan sistem tersebut menjadi lebih *reliable* dan menjadi sistem yang *fault tolerant*. Pada penelitian ini dilakukan implementasi metode *cold standby redundancy* pada objek sistem kunci otomatis serta memodelkan dan menganalisis hasil peningkatan *reliability* yang dihasilkan.

Perbandingan penelitian – penelitian sebelumnya dengan penelitian yang dilakukan dirumuskan dalam tabel yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 Perbandingan tinjauan pustaka

No	Nama Penulis [Tahun], Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Roosano & Purnomo[2016], Desain dan Prototipe Kunci Pintu Otomatis Menggunakan RFID Berbasis Arduino Uno	Menggunakan board mikrokontroller yang sama yaitu Arduino Uno.	Tidak menggunakan redundansi sebagai *backup component*	Ditambahkan redundansi pada salah satu komponen penting yaitu Arduino Uno
			Tidak dilakukan pengujian terhadap kegagalan sistem	Dilakukan pengujian dan analisis terhadap *reliability* sistem
2	Saputro & Wibawanto[2016], Rancang Bangun Pengaman Pintu Otomatis Menggunakan *E-KTP* Berbasis Mikrokontroler ATmega 328	Menggunakan sistem yang sama yaitu RFID sebagai input, solenoid sebagai *actuator* dan ATmega 328 sebagai pemroses.	Menggunakan mikrokontroler yang berdiri sendiri dengan rangkaian manual.	Menggunakan mikrokontroler yang sudah dalam bentuk *board* pabrikan yaitu Arduino Uno
			Tidak terdapat *backup component* dan juga pengujian terhadap kegagalan sistem.	Ditambahkan backup component dan dilakukan pengujian dan analisis terhadap *reliability* sistem.

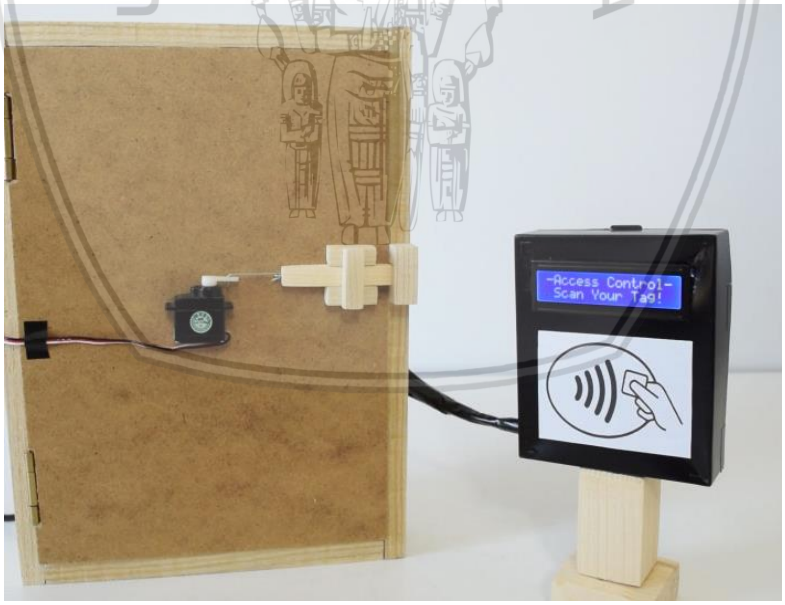
2.2 Dasar Teori

Dasar teori berisi tentang berbagai teori dasar yang dibutuhkan selama penelitian. Teori – teori dasar yang telah dikumpulkan dijelaskan pada sub bab ini. Beberapa diantaranya dasar teori tentang kunci pintu otomatis, *fault tolerant*, *reliability*, *hardware redundancy*, Arduino beserta komunikasi dan *library* yang digunakan.

2.2.1 Kunci Pintu Otomatis

Kunci pintu otomatis merupakan salah satu teknologi rumah pintar dalam bidang keamanan. Terdapat banyak sekali terobosan – terobosan teknologi di bidang keamanan terutama kunci pintu. Mulai dari yang berbasis kata sandi, sidik jari, RFID (Radio Frequency Identification), bahkan yang suara (*voice recognition*).

Diantara beberapa teknologi tersebut yang paling banyak dikembangkan saat ini adalah kunci pintu otomatis berbasi RFID. Teknologi RFID (*Radio Frequency Identification*) memnafaatkan medan elektromagnetik untuk transfer data untuk mengambil informasi dari RFID tag yang umumnya berbentuk kartu (Mathew & Divya, 2017). Kunci pintu berbasis RFID bekerja dengan menggunakan sejenis kartu yang memiliki ID tertentu yang dapat dibaca oleh RFID reader yang kemudian diproses oleh kontroler yang menggerakkan actuator tuas pengunci pintu. Contoh sistem kunci pintu otomatis berbasis RFID dapat dilihat pada Gambar 2.1.



Gambar 2.1 Kunci pintu otomatis berbasis RFID

Sumber : Nedelkovski (2017)

2.2.2 *Fault Tolerant*

Fault tolerant merupakan kemampuan sistem untuk tetap beroperasi dan menjalankan fungsi – fungsinya ketika terdapat kesalahan sistem (Johnson, 1984). Fault tolerance memiliki keterkaitan dengan *reliability* atau seberapa besar kemungkinan sebuah sistem dapat beroperasi dengan lancar tanpa adanya kerusakan yang mengakibatkan sistem tidak berfungsi. Sebuah sistem yang fault tolerant harus dapat mengatasi kegagalan – kegagalan yang terjadi pada masing – masing komponen hardware maupun software. Fault tolerant cukup penting karena tidak ada sistem yang sempurna, setiap sistem pasti memiliki beberapa kekurangan termasuk yang menyebabkan kegagalan sistem.

Dalam beberapa tahun terakhir, fokus dari fault tolerance beralih dari paradigma pengolahan informasi dengan komputer desktop tradisional dimana melibatkan user dalam dalam sebuah perangkat individu untuk tujuan tertentu, menjadi *ubiquitous computing* atau komputer yang bisa beroperasi dimana saja dengan ukuran yang kecil, perangkat – perangkat yang diproses secara bersamaan melalui sebuah jaringan dengan harga terjangkau dan dapat didistribusikan kedalam semua skala termasuk dalam kehidupan sehari – hari (Weiser, 1993). Dari pernyataan tersebut mungkin saat ini sudah sering kita jumpai perangkat – perangkat cerdas yang bersifat *ubiquitous computing* seperti perangkat *smart home*, *smart wearable device,* dan lain – lain yang dapat berfungsi tanpa perintah secara langsung dari penggunanya. Dan saat ini fault tolerance berfokus pada perangkat *ubiquitous computing* tersebut.

2.2.3 *Reliability*

Sebuah sistem yang *dependable* (bisa diandalkan) merupakan tujuan utama dari adanya pengembangan fault tolerance. Dalam artian luas, *dependability* merupakan kemampuan sistem untuk memberikan layanan kepada user (Lapric, 1985). Mengingat saat ini menginjak era *ubiquitous computing,* dependability berperan penting dalam pengembangan sistem tidak hanya dalam bidang keamanan, bisnis, dan sistem yang kompleks saja, namun saat ini merambah ke perangkat – perangkat dengan komputasi sederhana seperti *smart device* yang digunakan dalam kehidupan sehari – hari.

Reliability atau kehandalan merupakan salah satu atribut utama dari *dependability system. Reliability* atau jika disimbolkan menjadi *R(t)* merupakan probabilitas jalannya sebuah sistem tanpa adanya kegagalan dalam interval waktu tertentu atau [0, *t*], dimana sistem mulai beroperasi pada waktu ke 0 (Dubrova, 2013). Atau dalam arti lain, berapa lamakah sistem dapat berjalan mulai dari saat pertama kali beroperasi hingga menemui kegagalan pertama.

Reliability adalah ukuran dari layanan yang diberikan secara terus menerus. *High-Reliability* dibutuhkan ketika menginginkan sistem beroperasi secara normal tanpa adanya gangguan, seperti sistem tidak dapat beroperasi karena dalam perbaikan. Sebagai contoh, sistem kontrol pesawat dalam misi ruang angkasa diharapkan bisa beroperasi tanpa gangguan. Adanya cacat di sistem

ini kemungkinan akan menyebabkan kehancuran pesawat, seperti yang terjadi pada peluncuran Lewis spacecraft milik NASA pada tanggal 23 agustus 1997 (NASA, 2000). Pesawat ruang angkasa tersebut memasuki putaran di orbit yang mengakibatkan hilangnya tenaga surya dan pelepasan baterai yang fatal. Kontak dengan pesawat hilang dan pesawat ini hancur pada tanggal 28 September 1997. Menurut laporan investigasi gagalnya misi Lewis Spacecraft, kegagalan tersebut terjadi karena rancangan sistem *attitude-control* secara teknis sudah cacat dan pesawat ruang angkasa tersebut tidak dipantau secara memadai selama tahap awal operasinya.

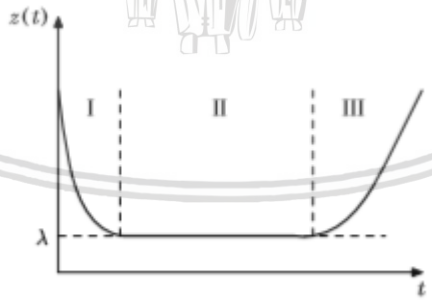
Reliability adalah sebuah fungsi waktu. Solusi dimana waktu spesifik pada variasi secara substansial berdasarkan pertimbangan dari sifat atau karakteristik sistem. *Reliability* menunjukkan probabilitas sukses dari sebuah sistem. Sebagai alternatif, kita dapat mendefinisikan *unreliability* atau $Q(t)$ dari sebuah sistem pada waktu t sebagai probabilitas sistem mengalami kegagalan pada interval [0, t], dimana sistem mulai beroperasi pada waktu 0. *Unreliability* menunjukkan probabilitas dari kegagalan dari sebuah sistem. Hubungan antara *reliability* dengan *unreliability* dapat di tunjukan pada persamaan berikut.

$$Q(t) = 1 - R(t) \tag{2.1}$$

2.2.4 Teknik Evaluasi *Reliability*

2.2.4.1 Failure Rate

Failure rate adalah ekspektasi jumlah kegagalan dalam satuan waktu tertentu (Siewiorek & Swarz, 1998). Umumnya sebuah hardware memiliki tiga fase evolusi yang secara grafik akan membentuk bathub curve seperti pada Gambar 2.2.



Gambar 2.2 Evolusi failure rate sebuah hardware

Sumber : Dubrova (2013)

Berdasarkan grafik pada Gambar 2.2 fase I adalah *infant mortality* dimana pada awal penggunaan *failure rate* dari *hardware* akan menurun secara signifikan yang biasanya disebabkan oleh kecacatan manufaktur. Setelah beberapa waktu *failure rate* dari *hardware* akan akan bersifat konstan, ini merupakan fase II yang disebut fase *useful life.* Fase III adalah kondisi *wear out* dari dari hardware dimana *failure rate* akan terus meningkat signifikan. Karena bersifat konstan *reliability*

sistem akan berkurang secara eksponensial berdasarkan waktu. Sehingga diperoleh persamaan berikut.

$$R(t) = e^{-\lambda t} \quad (2.2)$$

2.2.4.2 Reliability Block Diagram

Teknik evaluasi yang umum digunakan untuk menganalisis *reliability* sistem adalah dengan memodelkan sistem pada *Reliability Block Diagram* (RBD). RBD menampilkan sistem secara abstrak dalam blok-blok yang merepresentasikan komponen – komponen dalam sistem (Dubrova, 2013). Hubungan antar blok dalam RBD menggambarkan *dependency* antar komponen dalam sistem ketika beroperasi.

Gambar 2.3 *Reliability* Block Diagram Sistem Seri (kiri) dan Sistem Paralel (kanan)

Sumber : Dubrova (2013)

Reliability Block Diagram sebuah sistem dibagi menjadi dua yaitu sistem seri seperti pada Gambar 2.3 (kiri) dan sistem parallel pada Gambar 2.3 (kanan). Namun bisa juga sebuah sistem terdiri dari subsistem seri dan parallel yang tergabung menjadi satu.

Sistem seri dapat bekerja jika dan hanya jika semua komponen berfungsi dengan baik. Apabila salah satu komponen tidak bekerja maka sistem akan mengalami kegagalan. Jika masing – masing komponen memiliki nilai *reliability* yang independen maka persamaan *reliability* dari sistem seri adalah sebagai berikut.

$$R(t) = \prod_{i=1}^{n} Ci(t) \quad (2.3)$$

Persamaan tersebut menjelaskan bahwa *reliability* sistem didapat dari rangkaian perkalian dari nilai *reliability* pada setiap komponen *Ci* yang saling bergantung satu sama lain. Sedangkan sistem paralel untuk bisa bekerja membutuhkan setidaknya satu komponen bekerja. Sistem paralel memungkinkan adanya cadangan atau yang sering disebut dengan redundansi. Jika masing – masing komponen memiliki nilai *reliability* yang independen maka persamaan *reliability* dari sistem paralel adalah sebagai berikut

$$R(t) = 1 - \prod_{i=1}^{n}(1 - Ci(t)) \quad (2.4)$$

Persamaan tersebut memperlihatkan tidak adanya ketergantungan pada setiap komponen. *Reliability* sistem didapat dari *reliability* penuh dikurangkan rangkaian perkalian dari un*reliability* setiap komponen. Dengan menerapkan sistem paralel ini akan dapat meningkatkan *reliability* sistem.

2.2.5 Hardware Redundancy

Redundansi merupakan salah satu metode yang umum digunakan untuk meningkatkan kehandalan (*reliability*) dan ketersersediaan (*availability*) pada sebuah sistem. Menambahkan redundasi kedalam sistem tentunya akan menambah komleksitas dan biaya yang dikeluarkan, namun itu cukup sebanding dengan kehandalan sistem ketika ada suatu kegagalan operasi.

Hardare Redundancy adalah teknik redundansi yang dilakukan pada level hardware atau komponen fisik (Dubrova, 2013). Sebagai contoh sistem – sistem kompleks biasanya memiliki dua *processor*, dua *memory*, maupun dua *power supply*. Sistem tersebut menerapkan hardware redundancy dengan menambahkan jumlah *processor*, *memory*, ataupun *power supply*.

Hardware redundancy ada tiga jenis, yaitu *passive*, *active*, dan *hybrid*. *Passive redundancy* mewujudkan *fault tolerance* dengan cara menutupi kesalahan yang terjadi tanpa harus melakukan *action* apapun. *Active Redundancy* memerlukan pendeteksi kesalahan untuk mentolerir kesalahan yang terjadi. Setelah kesalahan terdeteksi sistem dapat memutuskan action apa yang akan dilakukan untuk menangani kegagalan tersebut. *Active redundancy* memerlukan waktu untuk rekonfigurasi yang biasa disebut *downtime*. *Hybrid redundancy* merupakan gabungan dari pendekatan *passive* dan *active redundancy. Hybrid redundancy* menggunakan mekanisme *masking* (menutupi kesalahan) serta *fault detector* (mendeteksi kesalahan) untuk menanggulangi kegagalan dan menggantinya dengan perangkat cadangan. *Hybrid redundancy* memungkinkan rekonfigurasi sistem tanpa adanya *downtime*.

Berdasarkan kriteria dari perangkat sistem yang ada, *active redundancy* merupakan metode yang paling cocok digunakan. *Active redundancy* dalam implementasinya memiliki beberapa metode, teknik, dan terminology. Terdapat tiga model utama yang umum digunakan dalam dunia industri. Diantaranya adalah *Standby Redundancy*, *N Modular Redundancy*, dan *1:N Redundancy*.

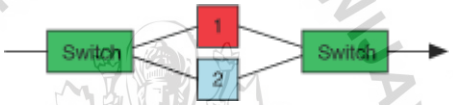
1. *Standby Redundancy*

   Standby Redundancy atau bisa disebut Backup Redundancy adalah ketika sebuah sistem memiliki unit sekunder yang identik dengan unit primer (National Instruments, 2008). Unit Sekunder secara typical tidak memonitor sistem, namun hanya bersifat cadangan. Unit sekunder atau unit standby umumnya tidak tersinkronisasi dengan unit primer, maka harus dicocokkan input dan output sinyalnya ketika mengambil alih dari Device Under Control (DUC). Pendekatan ini memiliki resiko kemungkinan terjadi *"bump"* pada transfer, yang berarti sekunder dapat mengirimkan sinyal kontrol ke DUC yang tidak selaras dengan sinyal kontrol terakhir yang berasal dari unit utama.

Mekanisme ini memerlukan pihak ketiga untuk menjadi pengawas, yang memonitor sistem untuk memutuskan kapan kondisi peralihan terpenuhi dan memerintahkan sistem untuk mengalihkan kontrol ke unit stanby dan voter, yang merupakan komponen yang memutuskan kapan harus beralih dan mana unit diberi kontrol dari DUC. Kenaikan biaya sistem untuk jenis redundansi ini biasanya sekitar 2X atau kurang tergantung pada biaya pengembangan perangkat lunak Anda. Dalam redundansi Standby ada dua tipe dasar yaitu *Cold Standby* dan Hot Standby.

a. Cold Standby

Dalam *Cold Standby*, unit sekunder dimatikan, sehingga keandalan unit terjaga. Kelemahan dari desain ini adalah downtime lebih besar daripada hot standby, karena kita harus mengaktifkan unit standby sampai pada *state* yang diketahui. Hal ini akan memberi tantangan dalam masalah sinkronisasi. Jeda waktu untuk membawa unit standby ke *online state* yang cukup lama kemungkinan akan berdampak cukup besar pada saat peralihan. Blok diagram sistem *Cold Standby* bisa dilihat pada Gambar 2.4.

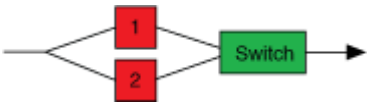


Gambar 2.4 Blok diagram *Cold Standby*

Sumber : National Instruments (2008)

b. Hot Standby

Dalam hot standby, unit sekunder dalam kondisi aktif dan secara opsional dapat memantau DUC. Jika menggunakan unit sekunder sebagai pengawas atau voter untuk memutuskan kapan harus beralih, Anda dapat menghilangkan kebutuhan pihak ketiga untuk jenis *hot standby* ini ini. Desain ini tidak menjaga keandalan unit *standby* seperti desain *cold standby*. Namun, *downtime* yang ditimbulkan lebih pendek sehingga meningkatkan ketersediaan sistem. Blok diagram sistem *hot standby* bisa dilihat pada Gambar 2.5.



Gambar 2.5 Blok diagram *hot standby*

Sumber : National Instruments (2008)

2.2.6 Arduino

Arduino adalah platform elektronik yang open-source berbasis hardware dan software yang mudah digunakan. *Board* Arduino dapat membaca input dari berbagai macam sensor seperti cahaya, sidik jari, bahkan pesan twitter lalu memprosesnya menjadi output untuk mengaktifkan actuator sepeti motor, lampu LED, dan lain – lain. *Board* Arduino dapat di beri perintah dengan memberikan satu set perintah pada microcontroller yang ada pada *board* Arduino. Untuk melakukannya dapat menggunakan Arduino programming language dan Arduino Software (IDE) (Arduino.cc, 2017) .

Arduino lahir di *Ivrea Interaction Design Institute* sebagai alat yang mudah digunakan untuk pembuatan prototipe dengan cepat. Arduino ditujukan untuk siswa yang tidak memiliki latar belakang dalam bidang elektronika dan pemrograman. Begitu sampai pada komunitas yang lebih luas, *board* Arduino mulai dikembangkan untuk menyesuaikan diri dengan kebutuhan dan tantangan baru, dari papan 8-bit sederhana hingga produk untuk aplikasi IoT, 3D painting, dan *embedded environment*. Semua *board* Arduino benar-benar open-source, memberdayakan pengguna untuk membangunnya secara mandiri dan menyesuaikannya dengan kebutuhan khusus mereka. Perangkat lunak juga open source, dan berkembang melalui kontribusi pengguna di seluruh dunia.

2.2.7 Arduino Uno

Arduino Uno adalah *developer board* Arduino berbasis mikrokontroler *ATmega328p*. Mikrokontroler *ATmega328p* memiliki 14 pin *digital* I/O dengan 6 diantaranya dapat digunakan sebagai output PWM. Memiliki 6 pin *analog input*, 16 MHz *crystal clock*, *USB conector*, *power jack*, *ICSP header* dan tombol *reset*.

"Uno" berarti satu dalam bahasa Italia dan dipilih sebagai tanda perilisan Arduino Software (IDE) 1.0. *Board* Uno dan Arduino Software (IDE) versi 1.0 adalah versi referensi Arduino, sekarang berevolusi ke rilis yang lebih baru. *board* Uno adalah yang pertama dari rangkaian *board* Arduino yang menggunakan USB, dan model referensi untuk platform Arduino (Arduino.cc, 2017). *Board* Arduino Uno dapat dilihat pada Gambar 2.6.

Gambar 2.6 *Board* Arduino Uno

Sumber: Arduino.cc (2017)

2.2.7.1 Spesifikasi

Spesifikasi lengkap dari *board* Arduino Uno dapat dilihat pada Tabel 2.2.

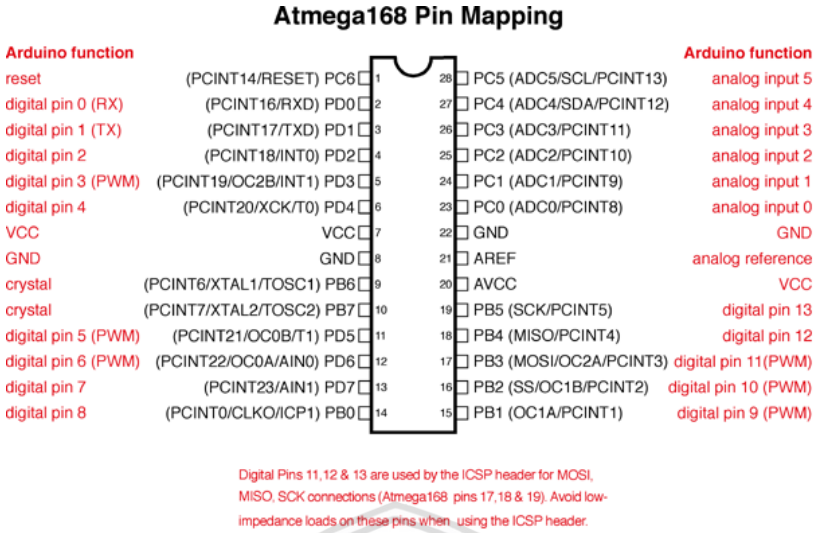
Tabel 2.2 Spesifikasi *Board* Arduino Uno

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Sumber: Arduino.cc (2017)

2.2.7.2 Pin Mapping

Board Arduino Uno menggunakan mikrokontroler *ATmega328p* dengan pemetaan pin seperti pada Gambar 2.7.



Gambar 2.7 Pemetaan pin ATmega168 / ATmega328

Sumber: Arduino.cc (2017)

2.2.7.3 Reliability Arduino Uno

Arduino tidak mempublikasikan nilai *reliability* dari produk – produknya termasuk *board* Arduino Uno. Arduino hanya memberikan informasi garansi 2 tahun untuk pembelian di Eropa dan 1 tahun untuk pembelian diluar Eropa (Arduino.cc, 2017). Untuk melakukan evaluasi *reliability* sistem berbasis Arduino tentu membutuhkan nilai *reliability* dari *board* Arduino itu sendiri. Setelah melakukan beberapa penulusuran melalui internet ditemukan artikel yang ditulis oleh seorang *professional engineer* bernama George Novacek yang memprediksi nilai *failure rate* Arduino Uno dengan berbagai metode prediksi *reliability* dan komputasi program.

George menerapkan beberapa metode perhitungan prediksi *reliability* untuk menghitung *failure rate* dari *board* Arduino Uno. Dalam melakukan prediksinya disebutkan bahwa untuk melakukan perhitungan prediksi reliability menggunakan beberapa parameter yang bersifat asumtif yaitu waktu operasi dan suhu ruangan tempat *board* Arduino beroperasi. Waktu operasi yang digunakan oleh George adalah 600.000 jam dan suhu ruangan antara -32 °C sampai 39 °C.

Salah satu hasil perhitungan *failure rate* akan dipilih untuk digunakan sebagai acuan dalam menghitung *reliability board* Arduino dalam penelitian ini. Salah satu nilai *failure rate* yang ambil adalah hasil perhitungan dari komputasi program komersil yang dibuat sendiri oleh George dengan metode kalkulasi prediksi reliability *Bellcore V6* (metode prediksi *reliability* untuk perangkat telekomunikasi). Hasil tersebut menunjukkan nilai failure rate 12.3357, jika *board* telah beroperasi selama 600.000 jam seperi yang telah disebutkan maka nilai failure rate per jam adalah 2.0559×10^{-5}. Hasil ini bukan merupakan nilai *failure rate* pasti dari Arduino Uno, namun nilai ini hanya prediksi yang dilakukan oleh George Novacek untuk digunakan dalam analisis *reliability* pada penlitian ini.

2.2.8 Arduino Nano

Arduino Nano merupakan *board* Arduino yang memiliki kemiripan dengan Arduino Uno, yaitu sama-sama menggunakan mikrokontroler ATmega 328 (Arduino nano versi 3.x). Namun *board* Nano memiliki ukuran yang lebih kecil daripada Arduino Uno. *Board* ini hanya dibekali sebuah Mini-B USB connector untuk menghubungkan *board* dengan PC dan juga sebagai catu daya. *Board* Arduino Nano dapat dilihat pada Gambar 2.8.

Gambar 2.8 *Board* Arduino Nano

Sumber : Arduino.cc (2017)

2.2.8.1 Spesifikasi

Spesifikasi lengkap dari *board* Arduino Nano dapat dilihat pada Tabel 2.3.

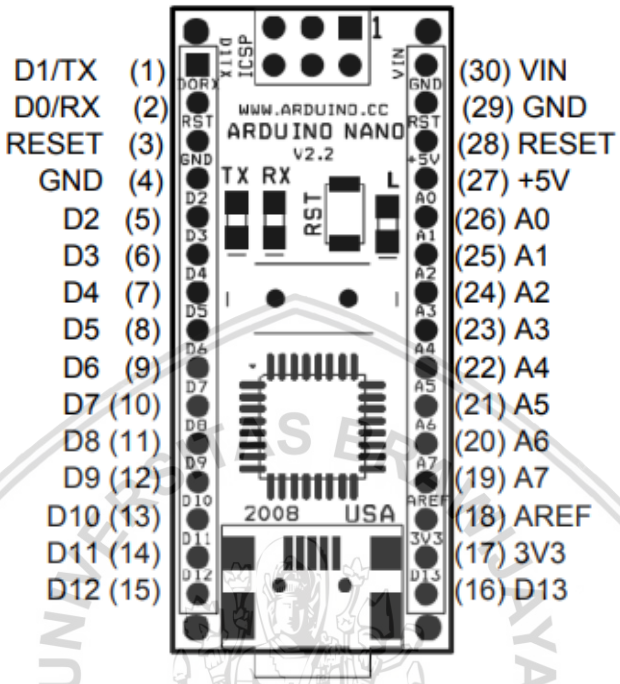
Tabel 2.3 Spesifikasi *Board* Arduino Nano

Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog I/O Pins	8
EEPROM	1 KB
DC Current per I/O Pin	40 mA (I/O Pins)
Input Voltage	7-12 V
Digital I/O Pins	22
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm
Weight	7 g
Product Code	A000005

Sumber : Arduino.cc (2017)

2.2.8.2 Pin Mapping

Board Arduino Nano menggunakan mikrokontroler *ATmega328* sehingga memiliki kemiripan pemetaan pin dengan Arduino Uno. Pemetaan pin Arduino Nano dapat dilihat pada Gambar 2.9.



Gambar 2.9 Pemetaan pin Arduino Nano

Sumber : Arduino.cc (2017)

2.2.9 Arduino Software (IDE)

Arduino Integrated Development Environment (IDE) adalah sebuah software dari platform Arduino yang terdiri dari editor teks untuk kode program, konsol teks, dan beberapa tombol – tombol toolbar dan menu (Arduino.cc, 2017).Software ini menghubungkan pengembang dengan perangkat keras Arduino untuk meng-upload program komunikasi serial USB. Software yang ditulis menggunakan Arduino disebut sketsa. Sketsa ini ditulis dalam editor teks. Sketsa disimpan dengan ekstensi file .ino. Editor ini memiliki fitur untuk meng-cut/paste dan untuk mencari / mengganti teks.

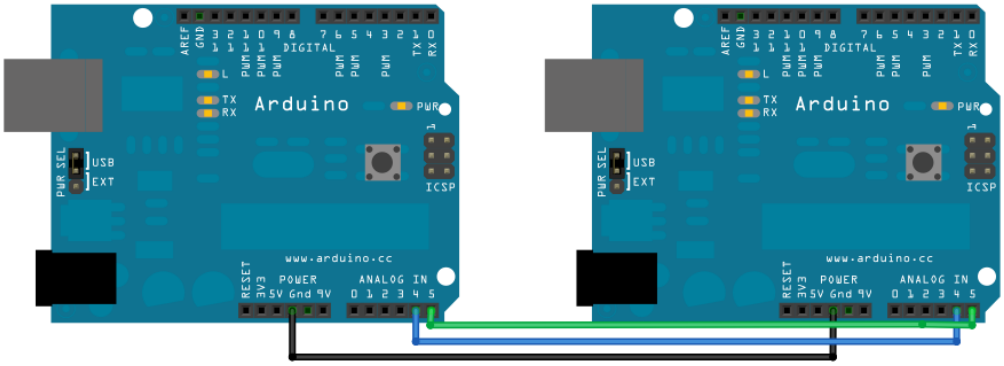
Dibagian bawah terdapat jendela pesan yang memberikan umpan balik sambil menyimpan dan mengekspor serta menampilkan kesalahan. Konsol menampilkan teks output termasuk pesan error dan informasi lainnya. Bagian bawah sebelah kanan sudut jendela menampilkan *board* yang digunakan saat ini dan port serial yang digunakan. Tombol-tombol toolbar memungkinkan Anda untuk memverifikasi dan mengunggah program, membuat, membuka, dan menyimpan sketsa, dan membuka serial monitor. Tampilan Arduino IDE dapat dilihat pada Gambar 2.10.

Gambar 2.10 *Sketch editor* Arduino IDE

sumber : IDE Arduino 1.8.0 (2017)

2.2.10 Komunikasi Serial I2C Arduino

Komunikasi serial saat ini sangat popular digunakan pada perangkat dengan komputasi rendah. Salah satu komunikasi serial Arduino yang popular adalah komunikasi I2C. Komunikasi I2C mentransmisikan data berdasarkan alamat unik dari perangkat. Komunikasi I2C menggunakan 4 pin yaitu vcc, ground, SDA (serial data), dan SCL (serial clock). Dari segi pemrograman I2C dapat digunakan dengan menggunakan *library* Wire. Untuk menghubungkan dua Arduino dengan komunikasi I2C cukup dengan menghubungkan pin ground, SDA, dan SCL dari masing – masing *board* Arduino. Konfigurasi pin hubungan antar dua Arduino melalui I2C dapat dilihat pada Gambar 2.11.



Gambar 2.11 Komunikasi I2C antara dua Arduino

Sumber: Arduino.cc (2017)

2.2.11 *Wire Library*

Wire library merupakan *library* dari Arduino yang memungkinkan komunikasi dengan perangkat I2C/TWI (Arduino.cc, 2017). *Library* ini memiliki banyak fungsi seperti *read(), write(), requestFrom() , onRequest(),* dan lain lain untuk mendukung komunikasi serial I2C.

Library ini menggunakan buffer sebesar 32 byte untuk menampung data selama komunikasi. Oleh karena itu setiap komunikasi yang berlangsung tidak boleh melebihi batas tersebut. Jika terdapat suatu transmisi data yang melebihi batas tersebut maka transmisi data tersebut akan langsung di drop.

2.2.12 *VirtualDelay Library*

fungsi *delay()* Arduino merupakan fungsi yang digunakan untuk memberikan jeda waktu pada program arduino. Namun untuk beberapa kasus implementasi fungsi *delay()* ini menjadi tidak efektif karena sifatnya yang menghentikan jalannya program selama jeda berlangsung. Hal ini membuat Arduino tidak bisa menjalankan program *multitasking* seperti membuat *delay* yang berbeda berjalan diwaktu yang bersamaan (Albert, 2017).

Melihat hal tersebut seorang developer bernama Albert van Dalen memutuskan untuk mengembangkan *VirtualDelay library* untuk Arduino yang dapat membuat fungsi jeda yang tidak mem-blok jalannya program. Berikut beberapa keuntungan menggunakan *VirtualDelay library.*

1. Selama *delay* berlangsung, program akan tetap berjalan
2. Dapat menggunakan beberapa *delay* secara sekuensial dalam satu kali *loop*.
3. Dapat menggunakan beberapa *delay* secara simultan dan independen.
4. Waktu *delay* bisa di set dalam mikrodetik atau milidetik
5. Tidak memerlukan perangkat keras tambahan
6. *Library* ini merupakan *timer-rollover* yang aman

Albert menyebutkan ada beberapa hal yang perlu diperhatikan dalam menggunakan *library* ini diantaranya adalah :

1. Fungsi *VirtualDelay* harus dijalankan didalam fungsi loop pada Arduino yang berjalan secara terus menerus.
2. Berbeda dengan fungsi *delay* standar, *VirtualDelay* membutuhkan fungsi *start()* dan *elapsed()* untuk mendandai waktu awal dan akhir jeda waktu.
3. Nilai untuk waktu jeda adalah diantara 0 sampai 2147483647

Untuk menggunakan *VirtualDelay* pada program tertentu cukup menyisipkan "avdweb_*VirtualDelay*.h" pada Arduino IDE dan sebuah variabel sebagai fungsi *milis*. Contoh sederhana dari penggunaan *VirtualDelay* dapat dilihat pada contoh program berikut.

Contoh program blinking LED dengan *VirtualDelay*

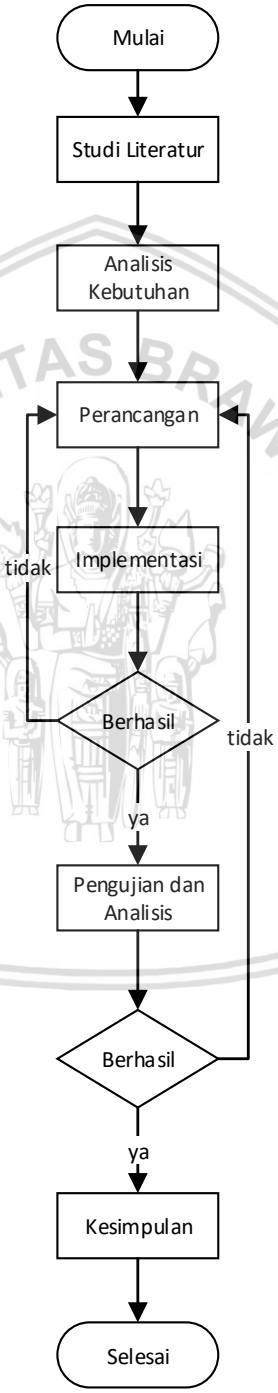
```
1  #include <Arduino.h>
2  #include "avdweb_VirtualDelay.h"
3
4  const byte ledPin = 13;
5  bool b;
6  VirtualDelay singleDelay;
7
8  void setup()
9  { pinMode(ledPin, OUTPUT);
10 }
11
12 void loop()
13 { singleDelay.start(400);
14   if(singleDelay.elapsed()) digitalWrite(ledPin, b=!b);
15 }
```

Sumber : Albert (2017)

Program tersebut merupakan program untuk membuat LED blink dengan *VirtualDelay*. Jeda waktu yang digunakan adalah 400 milisecond.

BAB 3 METODOLOGI

Terdapat beberapa langkah atau poin – poin dalam proses penelitian yang akan dilakukan. Langkah – langkah tersebut diilustrasikan melalui diagram alir untuk mempermudah dalam penelitian. Berikut diagram alir metodologi penelitian yang akan dilakukan.



Gambar 3.1 Diagram alir metode penelitian

3.1 Studi Literatur

Mempelajari berbagai referensi yang berkaitan dengan permasalahan yang diangkat serta sasaran atau teknologi yang digunakan. Sumber yang digunakan berupa text book, paper, skripsi, dan lain – lain. Referensi pendukung untuk pelaksanaan penelitian ini diantaranya adalah :

a. Kunci Pintu Otomatis
b. Fault Tolerant
c. *Reliability*
d. Harware Redundancy
e. Arduino

3.2 Analisis Kebutuhan

Analisis kebutuhan merupakan tahap yang penting untuk membangun sebuah sistem. Tahap ini dilakukan untuk dapat mengetahui apa saja yang dibutuhkan oleh sistem dari sisi fungsional maupun non fungsional. Hal ini juga sangat penting karena akan menentukan bagaimana spesifikasi dari sistem. Dengan adanya analisis kebutuhan ini diharapkan dapat mempermudah dalam perancangan dan implementasi serta pengujian sistem yang akan dilakukan. Berikut analisis kebutuhan untuk "Impelementasi Hardware Redundancy Pada *Switch*ing Pintu Otomatis Dengan Metode *Cold Standby* Dan Watchdog" :

3.2.1 Kebutuhan Perangkat keras

Beberapa perangkat keras dibutuhkan untuk membangun sistem redundansi dalam penelitian ini. Berikut perangkat keras yang dibutuhkan sistem:

1. Laptop / PC
2. Sistem Kunci Pintu Otomatis berbasis Arduino
3. Arduino Uno (sebagai redundant)
4. Arduino Nano (sebagai *fault detector*)
5. Relay module
6. Kabel Jumper

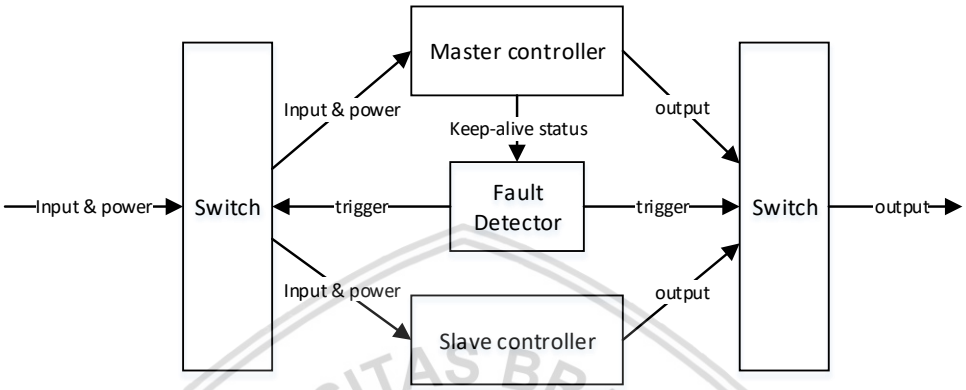
3.2.2 Kebutuhan Perangkat Lunak

Program dasar sistem kunci pintu otomatis perlu dimodifikasi agar dapat menjalankan sistem redundansi. Perangkat *fault detector* juga perlu diprogram untuk dapat menjadi watchdog. Untuk memenuhi kebutuhan tersebut dibutuhkan perangkat lunak sebagai editor sekaligus compiler kode program. Berikut kebutuhan perangkat lunak yang dibutuhkan sistem.

1. Arduino IDE
2. *Library* Wire (untuk komunikasi serial I2C)
3. *Library VirtualDelay* (untuk pengiriman *Keep-alive status*)

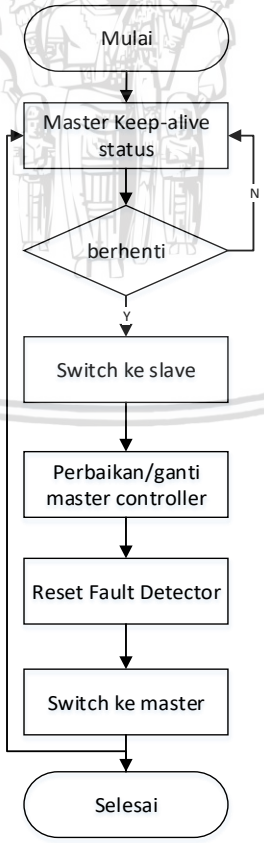
3.3 Perancangan Sistem

Tahap ini menjelaskan deskripsi umum tentang bagaimana sistem yang akan dikembangkan. Sistem ini dirancang dengan menggabungkan semua sub sistem pada analisis kebutuhan. Perancangan sistem akan dianalisis pada tiap – tiap sub sistem yang saling terhubung satu sama lain. Perancangan sistem digambarkan dalam bentuk blok diagram pada Gambar 3.2.



Gambar 3.2 Blok diagram sistem

Selain blok diagram sebagai desain hubungan antar komponen, juga dibuat diagram alir yang menunjukkan bagaimana sistem bekerja. Diagram alir sistem dapat dilihat pada Gambar 3.3.



Gambar 3.3 Diagram alir sistem

Diagram alir pada Gambar 3.3 menjelaskan bagaimana cara kerja sistem secara berurutan. Dengan demikian sistem lebih mudah dipelajari. Sehingga lebih mudah untuk mengevaluasi kesalahan pada fungsi sistem.

3.3.1 Perancangan Perangkat Keras

Berdasarkan Gambar 3.2 berikut detil analisis pada setiap komponen dan hubungan antar komponen :

1. *Master Controller* (Arduino Uno) dan *Fault detector* (Arduino Nano) terhubung melalui antarmuka komunikasi serial I2C dengan alamat 42.
2. Digital pin output dari *fault detector* terhubung dengan perangkat *switch* dimana seluruhnya merupakan *digital output trigger* untuk mengalihkan sistem ke *slave controller.*
3. Digital pin I/O *master controller* dan *slave controller* (Arduino Uno) terhubung dengan perangkat *Switch* (*Relay module*).
4. *Switch* (Relay Module) terhubung dengan *master controller, slave controller, fault detector* dan perangkat I/O sistem. Semua terhubung dengan antarmuka digital kecuali *power source*. *Relay module* yang digunakan masing 8 *channel* untuk input dan 4 *channel* untuk output.
5. Konfigurasi jalur relay *Normally Close* (NC) dipasangkan untuk *master controller*, sedangkan *Normally Open* (NO) dipasangkan untuk *slave controller*. Jalur COM dipasangkan untuk perangkat Input dan Output.

3.3.2 Perancangan Perangkat Lunak

Berdasarkan Gambar 3.2 berikut detil analisis pada setiap komponen dan hubungan antar komponen :

1. *Kode program master controller dimodifikasi* untuk mengirim *keep-alive status* kepada *fault detector* melalui komunikasi serial I2C dengan Fungsi *Wire.requestFrom().*
2. Fungsi *Wire.requestFrom()* dijalankan di dalam fungsi *VirtualDelay* dari *library VirtualDelay*. Tujuannya adalah agar pengiriman *keep-alive status* berjalan secara *multitask* dan tidak mengganggu jalannya program yang sudah ada.
3. *Keep-alive* dibaca berkala oleh *fault detector* menggunakan fungsi *Wire.onRequest()* dari *library wire.h* yang mana fungsi tersebut bekerja dengan fungsi berbasis *event* yaitu *requestEvent().*
4. *Slave controller* di berikan duplikat kode program dari *master* controller sebelum dimodifikasi.

3.4 Implementasi

Tahap selanjutnya adalah implementasi sistem. Pada tahap ini desain atau rancangan sistem redundansi yang telah dibuat akan di implementasikan. Secara umum implementasi dilakukan dengan menambahkan sistem redundansi pada sistem yang sudah ada yaitu sistem kunci pintu otomatis. Implementasi yang dilakukan meliputi implementasi perangkat keras dan implementasi perangkat lunak.

3.4.1 Implementasi Perangkat Keras

Implementasi perangkat keras dilakukan dengan melakukan instalasi perangkat keras sistem redundansi pada sistem kunci pintu otomatis. Beberapa perangkat keras sistem redundansi diantaranya adalah *slave* controller (Arduino Uno), Fault Detector (Arduino Nano) dan *Switch* (Relay Module). Beberapa perangkat keras tersebut dihubungkan dengan kabel jumper sesuai dengan perancangan yang telah dibuat.

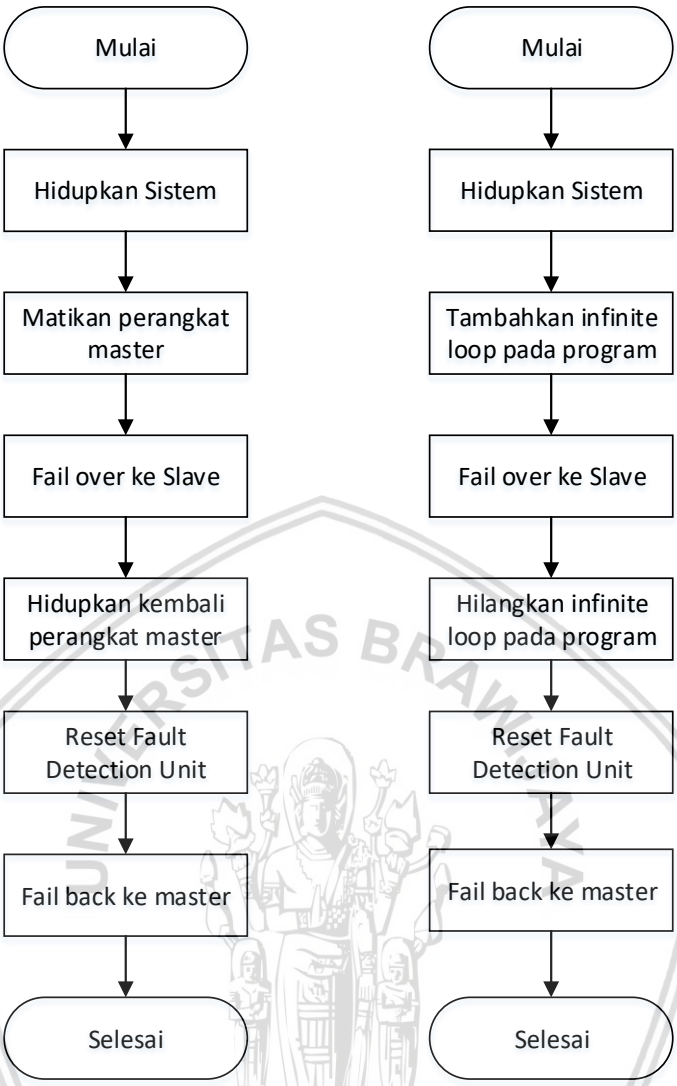
3.4.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak dilakukan dengan mengunggah kode program pada semua *board* mikrokontroler. Kode program untuk *master* controller merupakan modifikasi dari kode program awal dari sistem kunci pintu otomatis. Kode program untuk *slave* controller merupakan duplikat dari kode program sistem kunci pintu otomatis sebelum dimodifikasi. Kode program untuk *fault detector* merupakan kode penerimaan *keep-alive status* dan digital trigger untuk *switch* yang telah dibuat.

3.5 Pengujian dan Analisis

Pada tahap ini pengujian dilaksanakan secara eksperimental dan dilakukan berdasarkan kebutuhan fungsional dan non fungsional. Pengujian fungsional digunakan untuk mengetahui kerja sistem berdasarkan tujuan utama dari sistem itu sendiri. Sedangkan pengujian non fungsional dilakukan untuk mengetahui kinerja sistem selama beroperasi.

Pengujian keseluruhan sistem secara utuh dilakukan mengikuti skenario jalannya sistem dengan memberikan rekayasa kerusakan. Setelah kerusakan terjadi dilakukan analisis apakah sistem dapat mengatasi kegagalan dalam skenario. Terdapat dua skenario yang sudah dibuat untuk melakukan pengujian yaitu skenario secara hardware dan skenario secara software. Tujuannya adalah untuk mengetahui apakah sistem dapat mengatasi krusakan secara hardware maupun secara software. Skenario dibuat dalam bentuk diagram alir yang dapat dilihat pada Gambar 3.4.



Gambar 3.4 Diagram alir pengujian *switch*ing secara hardware (kiri) dan software (kanan)

Jika pengujian dalam skenario pada Gambar 3.3 berhasil tanpa ada kesalahan, maka sistem dapat dianalisis dengan metode evaluasi yang diperlukan. Namun jika pengujian mengalami kegagalan maka akan dilakukan evaluasi pada perancangan dan implementasi sistem. Setelah evaluasi pada perancangan dapat dilakukan perbaikan lalu kemudian dapat dilakukan pengujian kembali.

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah analisis dari hasil pengujian. Kesimpulan adalah hasil akhir yang didapatkan berdasarkan kesesuaian teori yang dikaji dan praktek pelaksanaan penelitian yang telah dilakukan. Selanjutnya penulisan saran – saran dari penulis untuk pengembangan atau penelitian lebih lanjut.

BAB 4 REKAYASA KEBUTUHAN

4.1 Perspektif Sistem

Tujuan dari sistem ini dapat dikatakan tercapai jika sistem mampu melakukan *fail over* dan *fail back* sesuai skenario pengujian. Sistem juga mampu melakukan *switch*ing sesuai keadaan *fault* yang ditentukan. Dapat membedakan antara *error* yang memerlukan *fail over* dengan *error* yang tidak memerlukan *fail over.*

4.2 Batasan Sistem

Berikut batasan – batasan dalam perancangan sistem:

1. Sistem menggunakan *Cold Standby Redundancy* untuk mekanisme *fail over* maupun *fail back*.
2. Sistem hanya menggunakan *single spare redundancy* dimana hanya ada satu *Redundant unit* dalam sistem.
3. *Fault detector* yang digunakan hanya menggunakan satu metode *watchdog*, yaitu *Keep-alive status.*
4. Menggunakan dua skenario untuk pengujian *switch*ing yaitu *power off* dan *infinite loop*.

4.3 Spesifikasi Sistem

Berdasarkan analisis kebutuhan yang telah dibuat, didapatkan rancangan spesifikasi sistem secara global. Pada dasarnya spesifikasi sistem ini merupakan tambahan dari spesifikasi awal sistem (sebelum menggunakan redundansi). Spesifikasi sistem yang akan dirancang adalah sebagai berikut:

1. Sistem memiliki dua kontroler yang bertindak sebagai *master* dan *slave*, dimana *slave* bertindak sebagai *backup* dari *master*.
2. Kontroler *slave* identik dengan *master controller* dengan I/O dan program yang sama dan mampu mengambil alih tugas *master controller*.
3. Sistem memiliki unit *fault detector* sebagi pendeteksi kesalahan yang terjadi pada *master controller* dan juga sebagai pendukung keputusan untuk *switching*. Dalam hal ini diasumsikan bahwa *fault detector* memiliki *reliability* yang lebih tinggi dari *master* maupun *slave controller*.
4. Sistem dapat melakukan *switch*ing untuk *fail over* ketika *master* mengalami kegagalan operasi dan juga *switch*ing untuk *fail back* keika *master* sudah diperbaiki dan dipasang kembali.
5. Sistem memiliki satu sumber listrik DC 9 volt sebagai catu daya untuk kedua kontroler dimana sumber listrik ini nanti juga akan *switch* oleh unit *switch* untuk memilih kontroler mana yang akan dihidupkan.

4.4 Analisis Kebutuhan

Setelah menentukan batasan dan spesifikasi sistem analisis kebutuhan dapat dilakukan berdasarkan fungsional dan non fungsional. Kebutuhan perangkat keras dan perangkat lunak yang sudah dipersiapkan sebelumnya akan dianalisis berdasarkan fungsi masing – masing. Hal ini dilakukan untuk mempermudah dalam perancangan, implementasi, serta pengujian sistem.

4.4.1 Kebutuhan Fungsional

Kebutuhan fungsional sistem merupakan fungsi keseluruhan sistem yang sesuai dengan tujuan dari sistem. Analisis kebutuhan fungsional dilakukan berdasarkan fungsi setiap tahap jalannya sistem. Secara umum fungsional sistem terdiri dari tiga bagian yaitu mengirim *keep-alive status,* membaca *keep-alive status,* dan *Switching*. *Keep-alive status* dikirimkan menggunakan fungsi komunikasi I2C *wire.RequestFrom* pada *library wire.h* pada Arduino IDE.

1. *Mengirim Keep-alive status (Master Controller)*

Saat pertama kali sistem dinyalakan *master controller* akan mengirimkan *Keep-alive status* kepada *fault detector* terus menerus. *Keep-alive status* yang dikirimkan merupakan tanda bahwa *master* controller masih bekerja. *Keep-alive status* harus dikirimkan secara multitask agar tidak mengganggu jalannya sistem kunci pintu otomatis.

2. *Membaca Keep-alive status (Fault Detector)*

Fault detector membaca *Keep-alive status* dari *master controller* sebagai acuan keputusan *switch*ing. Jika tidak menerima *keep-alive status* selama waktu tertentu maka *fault detector* akan mengirim trigger kepada *switch* device untuk melakukan *switch* ke *slave controller*.

3. *Mengalihkan ke Slave Controller (Switch)*

Switch bertugas mengalihkan sistem dari *master controller operate* menjadi *slave controller operate*. *Switch* bekerja berdasarkan *output trigger* dari sub sistem *fault detector.* Berikut detil analisis untuk kebutuhan mengalihkan ke *slave controller. Switch device* yang digunakan harus mampu melakukan *routing* untuk memilih salah satu diantara dua jalur sirkuit. Dalam hal ini device yang digunakan adalah *Relay module* karena memiliki kemampuan tersebut.

4.4.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional meliputi performa sistem dan akurasi kerja sistem pada setiap kebutuhan fungsionalnya. Kebutuhan non fungsional yang dapat diuraikan adalah kebutuhan mengenai akurasi pengiriman dan pembacaan *keep-alive status* dan jeda waktu selama *switch*ing. Berikut kebutuhan non fungsional sistem.

1. *Akurasi pengiriman dan pembacaan Keep-alive status*

Pengiriman dan pembacaan *Keep-alive status* harus dibuat seakurat mungkin dan selaras antara mikro*master controller* dengan *fault detector*. Karena hal ini merupakan acuan keputusan untuk melakukan *switch*ing. Mikro*master controller* mengirim *Keep-alive status* setiap 500ms kepada *fault detector*. Sedangkan Faul detector menunggu dan membaca setiap 500ms *Keep-alive status* dari mikro*master controller*.

2. *Jeda waktu peralihan master ke slave*

Peralihan dari mikro*master controller* ke *slave* diberikan waktu sebanyak 5 detik untuk memberikan pertimbangan kepada *fault detector* apakah error yang terjadi merupakan error yang bersifat sementara atau permanen.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini akan menjelaskan bagaimana proses perancangan sistem hingga implementasi sistem. Perancangan dan implementasi bertujuan untuk mengetahui bagaimana desain sistem dan prototype yang dapat dihasilkan. Perancangan dan implementasi dibuat secara detail pada setiap sub sistem untuk mempermudah dalam pengujian dan evaluasi sistem.

5.1 Perancangan Sistem

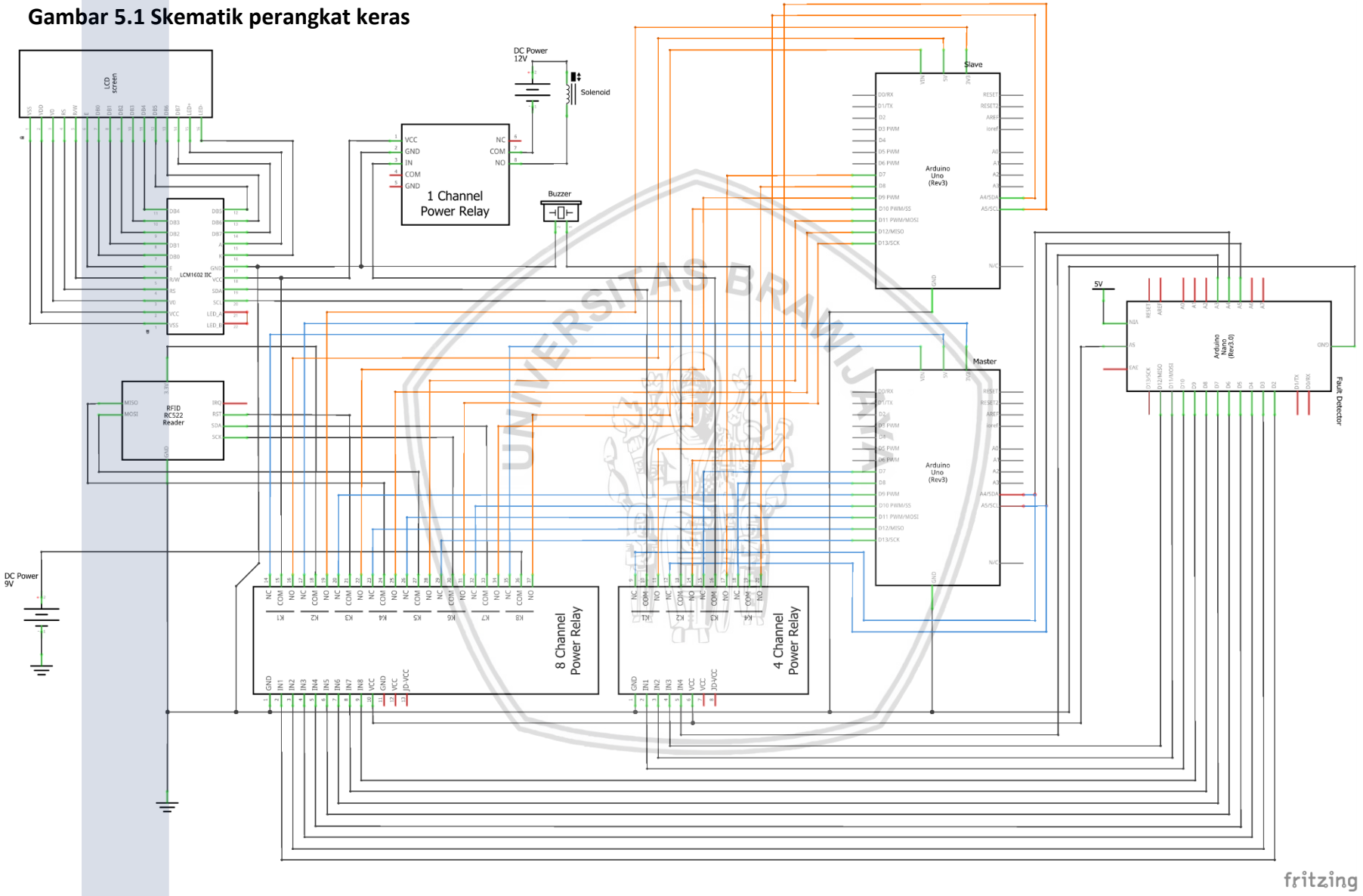
Perancangan sistem menjelaskan bagaimana rancangan atau desain yang akan diimplementasikan pada sistem. Pada bab ini akan dijelaskan bagaimana desain dan cara kerja pada tiap sub sistem sesuai spesifikasi yang ditentukan pada rekayasa kebutuhan. Dengan harapan untuk mengurangi kemungkinan kesalahan-kesalahan yang terjadi saat implementasi dan pengujian. Perancangan dibagi menjadi perancangan perangkat keras dan perancangan perangkat lunak.

5.1.1 Perancangan Perangkat Keras

Perancangan sistem yang telah dibuat dalam bentuk blok diagram pada Gambar 3.2 menggambarkan hubungan antar komponen. Blok diagram dibuat berdasarkan spesifikasi sistem yang telah dirancang. Dilihat dari penjelasan blok diagram tersebut terdapat beberapa perangkat keras yang saling terhubung.

Beberapa perangkat keras tersebut antara lain 2 *board* Arduino Uno, 1 *board* Arduino Nano, 2 relay module 8 channel dan 4 channel, perangkat Input dan Output (RFID, LCD, Buzzer, dan Solenoid yang dikontrol relay). Dari uraian tersebut berhasil digambarkan simulasi hubungan antar komponen secara riil melalui skematik diagram sistem yang dapat dilihat pada Gambar 5.1.

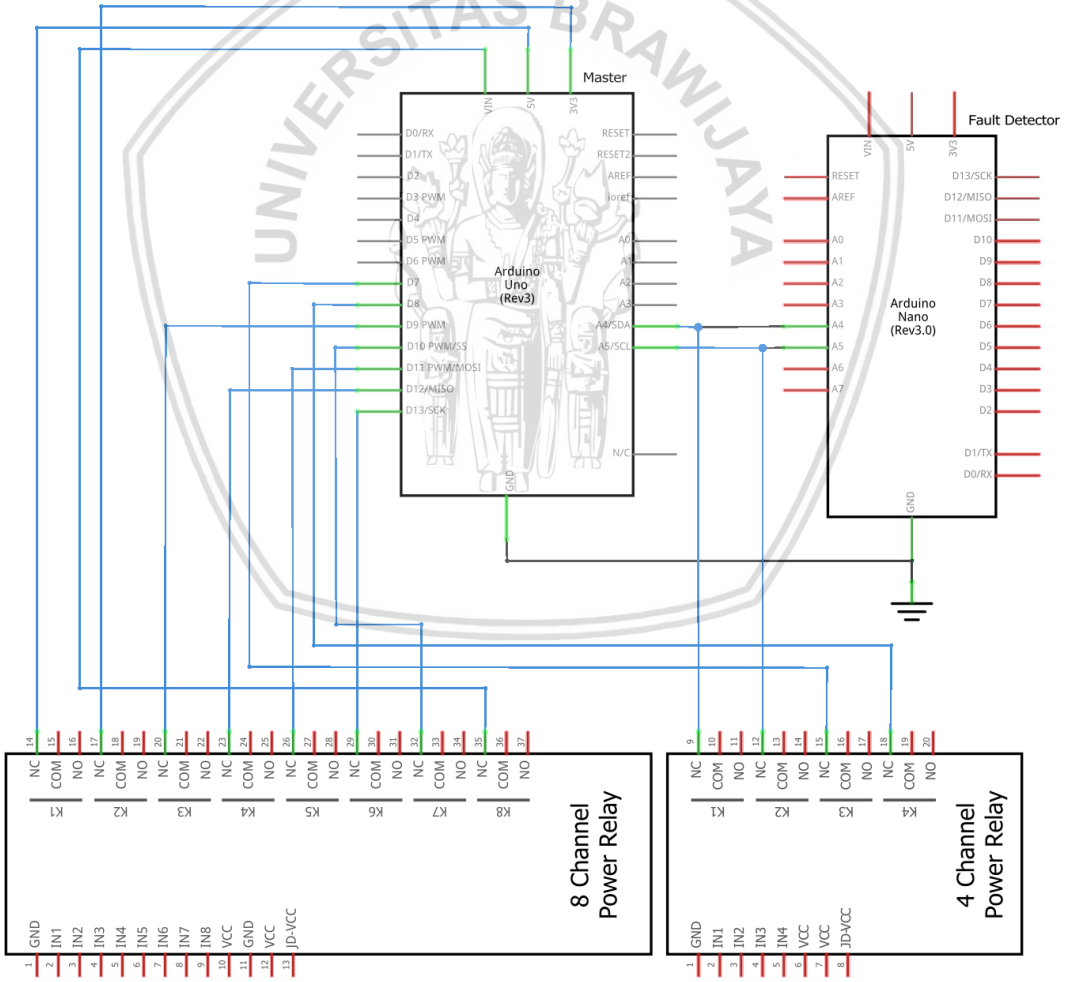
Gambar 5.1 Skematik perangkat keras



Berdasarkan skematik yang ada pada gambar 5.1 dapat diketahui hubungan antar masing – masing komponen yang satu dan lain. Pada skematik tersebut digunakan dua warna khusus untuk *wiring* I/O *master* dan *slave*. Warna biru untuk I/O dari *master* sedangkan warna oranye untuk I/O dari *slave*. Agar dapat lebih dipahami penjelasan skematik perancangan tersebut dipisah pada tiap – tiap komponen diantaranya *master controller, slave controller, fault detector, dan switch.*

5.1.1.1 Perancangan Master Controller

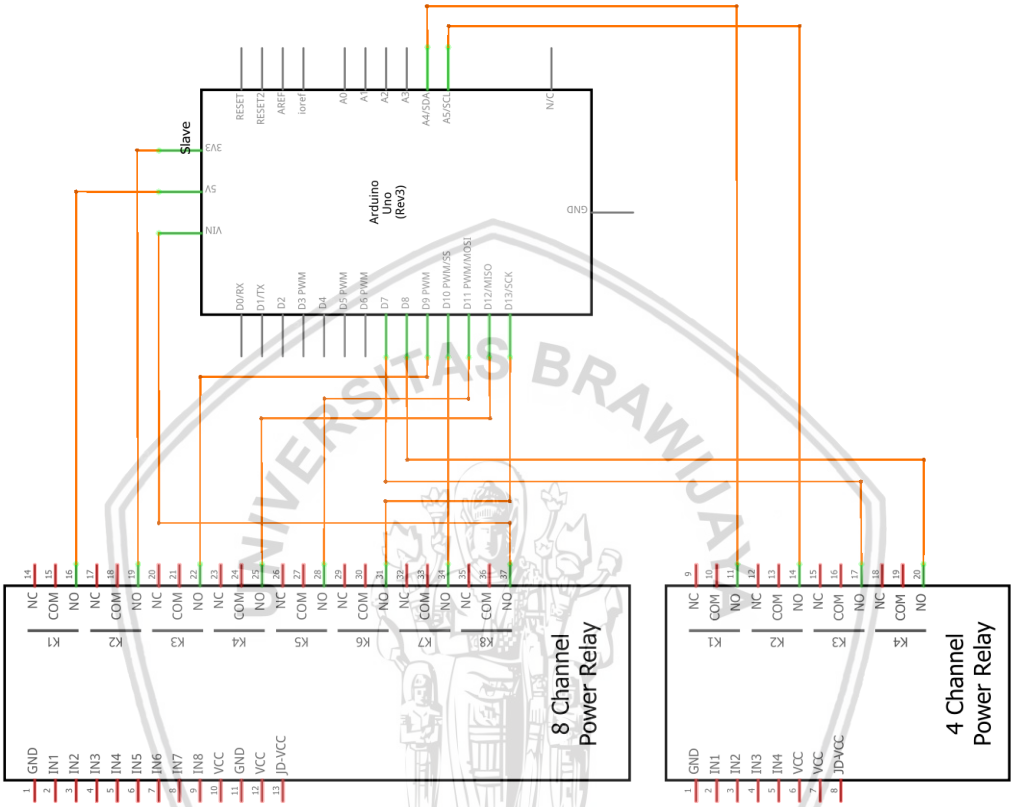
Master controller terhubung dengan *fault detector* dan *switch*. Perancangan ini terdiri dari perancangan hubungan antara Arduino Uno (*master*), Arduino Nano dan Relay module. Arduino Uno dan Arduino Nano terhubung dengan 3 pin yaitu pin GND (ground), pin A4 (SDA) dan pin A5 (SCL). Sedangkan Arduino Uno dan Relay module terhubung dengan 12 digital pin Input dan Output dari sistem kunci pintu otomatis pada port NC relay. Skematik diagram untuk perancangan *master controller* dapat dilihat pada Gambar 5.2.



Gambar 5.2 Skematik perancangan *master controller*

5.1.1.2 Perancangan Slave Controller

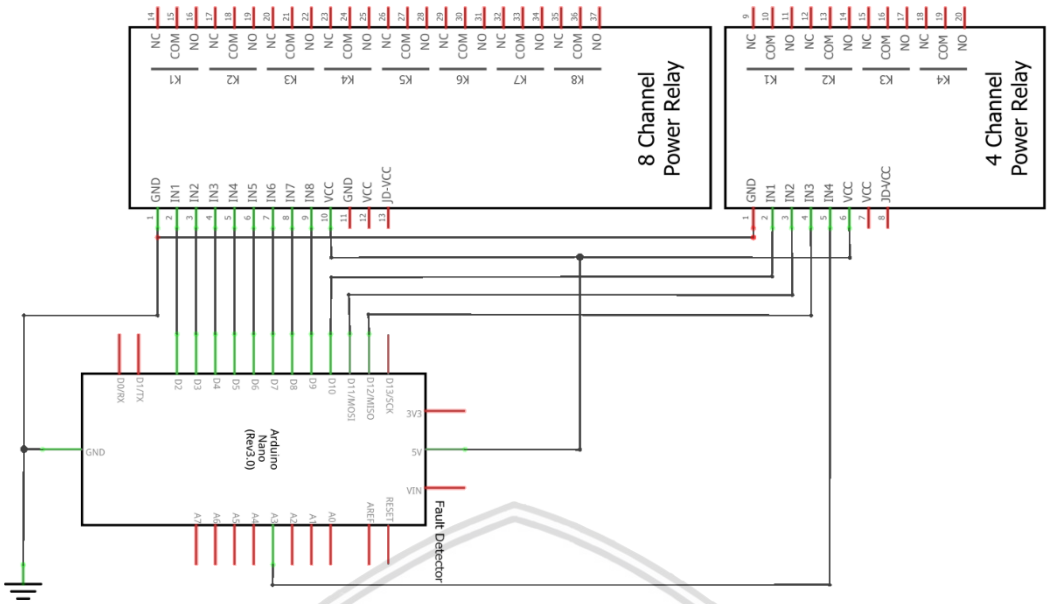
Slave controller hanya terhubung dengan *switch*. Perancangan ini terdiri dari perancangan hubungan antara Arduino Uno (*slave*) dan Relay module. Arduino Uno dan Relay module terhubung dengan 12 digital pin Input dan Output dari sistem kunci pintu otomatis pada port NO relay. Skematik diagram untuk perancangan *slave controller* dapat dilihat pada Gambar 5.3.



Gambar 5.3 Skematik perancangan *slave controller*

5.1.1.3 Perancangan Fault Detector

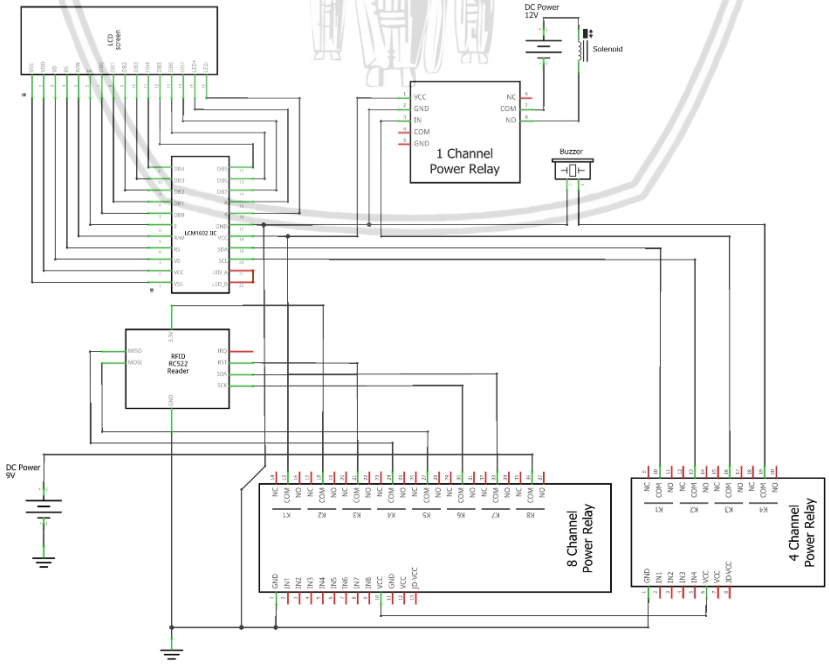
Fault detector terhubung dengan *switch dan master controller*. Hubungan *fault detector* dengan *master controller* sudah dapat dilihat pada Gambar 3.3. Oleh karena itu untuk perancangan ini hanya akan diperlihatkan perancangan hubungan antara Arduino Nano dengan Relay module. Keduanya terhubung dengan 12 pin digital input pada relay dan 12 output digital dari Arduino Nano (D2-D12 + A3 yang digunakan secara digital). Skematik diagram untuk perancangan *fault detector* dapat dilihat pada Gambar 5.4.



Gambar 5.4 Skematik perancangan *fault detector*

5.1.1.4 Perancangan Switch

Switch terhubung dengan *master controller, slave controller, fault detector,* dan perangkat Input dan Output sistem kunci pintu otomatis. Hubungan *switch* dengan *master controller* sudah dapat dilihat pada Gambar 5.2, *switch* dengan *slave controller* pada Gambar 5.3, dan *switch* dengan *fault detector* pada Gambar 5.4. Oleh karena itu untuk perancangan *switch* hanya akan diperlihatkan perancangan hubungan antara Relay module dengan perangkat Input dan Output sistem kunci pintu otomatis. Skematik diagram untuk perancangan *switch* dapat dilihat pada Gambar 5.5.



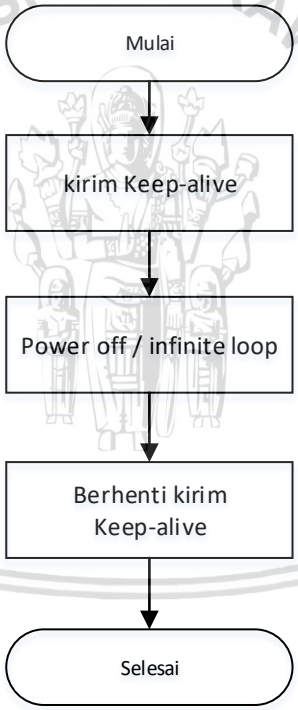
Gambar 5.5 Skematik perancangan *switch*

5.1.2 Perancangan Perangkat Lunak

Setelah membuat perancangan perangkat keras dilanjutkan pada perancangan perangkat lunak. Perancangan perangkat lunak dilakukan dengan membuat diagram alir pada tiap sub sistem berdasarkan diagram alir keseluruhan sistem yang ada pada Gambar 3.3. Membuat perancangan perangkat lunak pada sub sistem bertujuan untuk memudahkan dalam melakukan implementasi maupun evaluasi. Perancangan perangkat lunak dilakukan pada tiga sub sistem yaitu *Master Controller*, *Slave Controller,* dan *Fault detector* yang masing – masing memiliki fungsi yang berbeda.

5.1.2.1 Perancangan master controller

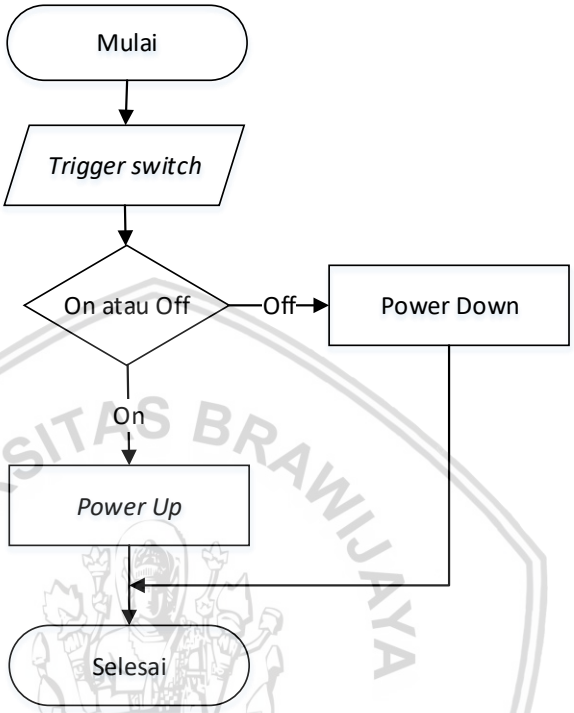
Master controller mengirimkan *keep-alive status* kepada *fault detector* seacara periodik. Pengiriman dilakukan menggunakan komunikasi serial I2C dan *VirtualDelay library* agar pengiriman berjalan secara *multitask*. Diagram alir *master controller* dapat dilihat pada Gambar 5.6.



Gambar 5.6 Diagram alir *master controller*

5.1.2.2 Perancangan slave controller

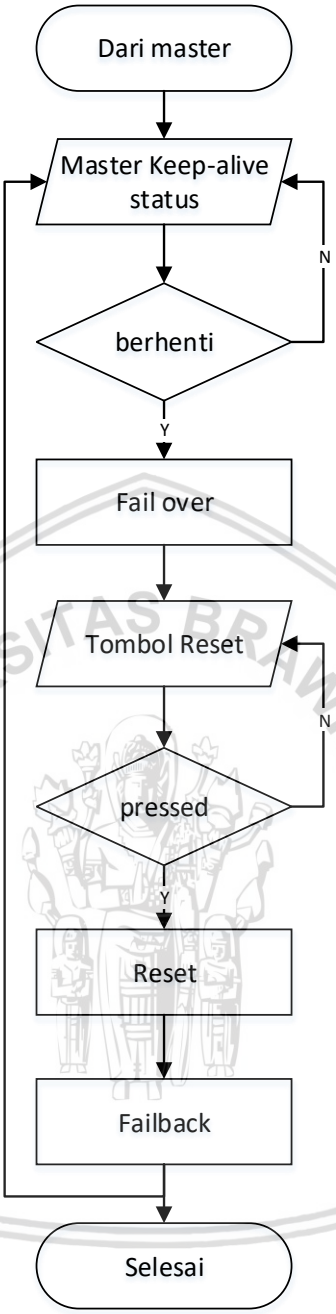
Slave controller bertugas sebagai pengganti ketika *master controller* mengalami kegagalan. *Slave* bekerja dibawah kendali *switch* yang mengalihkan catu daya serta I/O dari *master*. Cara kerja *slave* controller dapat dilihat pada gambar 5.7.



Gambar 5.7 Diagram alir *slave controller*

5.1.2.3 Perancangan fault detector

Fault detector atau *fault detector* bertugas mendeteksi kegagalan yang terjadi pada *master.* Unit ini bekerja berdasarkan *Keep-alive status* dari *master*. Sub sistem ini menjadi peran utama dari sistem yang mana *fault detector* merupakan jantung sistem *standby redundancy*. *Fault detector* harus mampu membedakan antara kegagalan yang memerlukan *fail over* atau tidak. Hal ini karena *Fault detector* juga bertanggung jawab atas keputusan *switch* modul untuk melakukan *fail over* maupun *fail back.* Cara kerja *Fault detector* dapat dilihat pada Gambar 5.8.



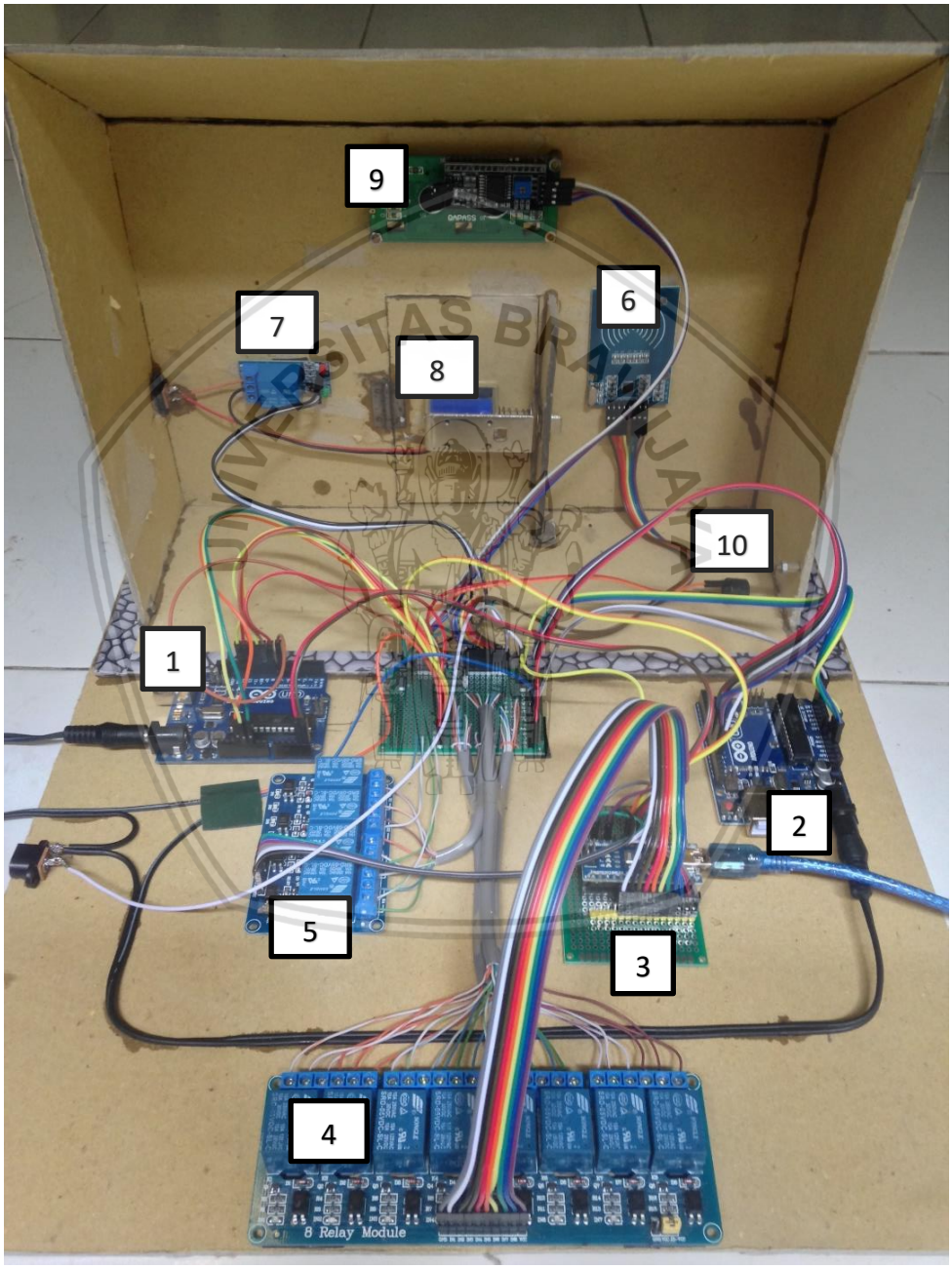
Gambar 5.8 Diagram alir *fault detector*

5.2 Implementasi Sistem

Bagian ini menjelaskan bagaimana proses implementasi dari desain dan perancangan sistem terhadap kebutuhan-kebutuhan yang sudah dikumpulkan. Karena penelitian ini bersifat pembangunan suatu alat, maka implementasi dikelompokkan dalam implementasi perangkat keras dan implementasi perangkat lunak.

5.2.1 Implementasi Perangkat keras

Implementasi perangkat keras dilakukan dengan mengimplementasikan rancangan skematik perangkat keras ke dalam prototype. Perangkat keras dari analisis kebutuhan di kumpulkan lalu dipasang sesuai konfigurasi hubungan pin antar perangkat sesuai skematik diagram. Perangkat keras saling dihubungkan dengan kabel – kabel jumper sesuai dengan skematik yang dibuat. Hasil implementasi perangkat keras dapat dilihat pada Gambar 5.9.



Gambar 5.9 Implementasi perangkat keras

Keterangan masing – masing komponen dari Gambar 5.9 :

1. Arduino Uno (*Master*)
2. Arduino Uno (*Slave*)
3. Arduino Nano (*Fault Detector*)
4. Relay 8 *channel* (*Switch* untuk *input*)
5. Relay 4 *channel* (*Switch* untuk *output*)
6. RFID *reader*
7. Relay 1 *channel* sebagai kontrol *solenoid*
8. *Solenoid door lock*
9. I2C LCD 16x2
10. Buzzer

5.2.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak dilakukan dengan upload kode program pada tiap mikrokontroler pada sub sistem *master*, *slave*, dan *fault detector* sesuai diagram alir masing – masing yang telah dijelaskan pada perancangan perangkat lunak. Implementasi perangkat lunak menjelaskan konfigurasi masing – masing sub sistem melalui cuplikan kode program yang terkait.

5.2.2.1 Konfigurasi master

Konfigurasi perangkat lunak *master* meliputi penambahan potongan kode program untuk mengirim sinyal *Keep-alive status* ke *fault detector*. Potongan kode program ini digunakan untuk mengirim data secara serial I2C menggunakan *library* wire.h pada Arduino IDE. Berikut cuplikan kode program dari pengiriman *Keep-alive status*.

Library dan variabel untuk konfigurasi *master* (selengkapnya pada Lampiran A.1)

```
1  #include <Wire.h>
2  #include < #include <avdweb_VirtualDelay.h>
3  size_t beat = 1;
4  VirtualDelay delayBeat;
```

Kode program konfigurasi *master* (selengkapnya pada Lampiran A.1)

```
1  void loop() {
2    delayBeat.start(500);
3    if(delayBeat.elapsed()){
4      byte returnedCount = Wire.requestFrom (42, beat);
5    }
6  }
```

Cuplikan kode program konfigurasi *master* ditambahkan dalam fungsi *void loop()* pada kode program *master*. Potongan kode program tersebut akan berjalan secara independen dan realtime. Oleh karena itu digunakan virtual non-blocking *delay* sebagai interval pengiriman tiap *Keep-alive status* untuk memenuhi kebutuhan tersebut. Virtual non-blocking *delay* merupakan *delay* yang memungkinkan proses lain tetap berjalan. Untuk menggunakan non-blocking *delay* ini dibutuhkan *VirtualDelay library* pada Arduino IDE yang cara penggunaannya bisa dilihat pada cuplikan kode program *Library* dan variabel untuk konfigurasi *master*.

5.2.2.2 Konfigurasi slave

Konfigurasi pada *slave* cukup dengan menduplikat dari kode program dari *master* controller. Kode program tersebut diupload pada redundant unit sebagai pengganti *master* controller. Namun kode program yang di duplikat adalah kode program yang masih belum ditambahkan kode pengiriman *Keep-alive status*. Hal ini karena proses *fail back* dilakukan secara manual dengan mereset perangkat *fault detector*. Dengan demikian kode program yang di upload ke *slave* tidak memerlukan *library VirtualDelay library* pada Arduino IDE.

5.2.2.3 Konfigurasi Fault detector

Konfigurasi *fault detector* merupakan konfigurasi yang paling utama dari sistem. Karena sub sistem ini bertanggung jawab atas jalannya sistem redundansi yang dibangun. Implementasi pada sub sistem ini meliputi kode untuk pembacaan *event Keep-alive status*, pengolahan *event* secara periodik, kemudian pemilihan keputusan untuk *fail over* dan *fail back*. Untuk semua fungsi tersebut diperlukan *library* sebagai berikut.

Library dan variabel untuk konfigurasi fault detector (selengkapnya pada Lampiran A.3)

```
1  #include <Wire.h>
2  bool x,y,z;
3  int faultCount = 0;
```

1. Pembacaan *event Keep-alive status*

Pembacaan *event Keep-alive status* menggunakan fungsi *requestEvent()*. Fungsi ini merupakan fungsi komunikasi serial I2C yang digunakan untuk membaca *event* yang masuk ke perangkat. Setiap *event* akan dibaca dan ditulis pada variabel Boolean dengan nilai yang saling bergantian setiap *event* datang. Berikut potongan kode program untuk menerima *event Keep-alive status* dari perangkat *master*.

Kode program untuk pembacaan *keep-alive status* (selengkapnya pada Lampiran A.3)

```
1  void requestEvent (){
2    Wire.write ("A", 1);
3    x = !x;
4    Serial.println("Master Status : ONLINE");
5    faultCount = 0;
6  }
```

2. Pengolahan *event* periodik

Pengolahan *event* yang dimaksud adalah memberikan nilai dari pembacaan fungsi *event Keep-alive status* pada dua variabel yang berbeda dan nantinya dapat digunakan untuk membandingkan apakah nilai variabel yang menyimpan *event* berubah atau tidak. Sedangkan periodik yang dimaksud adalah interval waktu yang diberikan saat pemberian nilai pada masing – masing variabel. Interval yang diberikan yaitu 500 milisecond sesuai dengan interval *event Keep-alive status*. Berikut potongan kode program pengolahan *event* periodik.

Kode program untuk pengolahan *event* periodik (selengkapnya pada Lampiran A.3)

```
1    y = x;
2    delay(500);
3    z = x;
4    delay(500);
```

3. Pemilihan keputusan *switch*ing

Pemilihan keputusan *switch*ing merupakan tahap akhir dari konfigurasi *fault detector*. Keputusan *switch*ing ini adalah penentuan apakah sistem akan melakukan *fail over* atau *fail back*. Keputusan didasarkan pada perubahan nilai dari variabel penyimpan *event* yang sudah diolah pada poin sebelumnya. jika nilai tidak mengalami perubahan selama lima waktu (lima kali pemberian nilai pada kedua variabel pengolahan *event*), maka perangkat *master* dianggap rusak atau mengalami kegagalan dan akan segera melakukan *fail over*. Berikut potongan kode program pemilihan keputusan *switch*ing.

Kode program untuk keputusan *switching* (selengkapnya pada Lampiran A.3)

```
1    if (y == z){
2      faultCount++;
3      if(faultCount > 5){
4        Serial.println("Master Status : OFFLINE");
5        relayLow();
6      }
7    }
```

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan pengujian terhadap rancangan dan implementasi yang telah dilakukan. Pengujian ini dilakukan untuk mengetahui apakah sistem sudah berjalan sesuai tujuan fungsional sistem dan mengetahui bagaimana performa serta akurasi sesuai tujuan non fungsional sistem. Setelah pengujian berhasil *reliability* sistem dianalisis untuk mengetahui serta membuktikan teori – teori yang terdapat pada landasan pustaka mengenai sistem redundansi. Sesuai pada rekayasa kebutuhan pengujian akan dilakukan berdasarkan fungsional dan non fungsional sistem.

6.1 Pengujian Fungsional

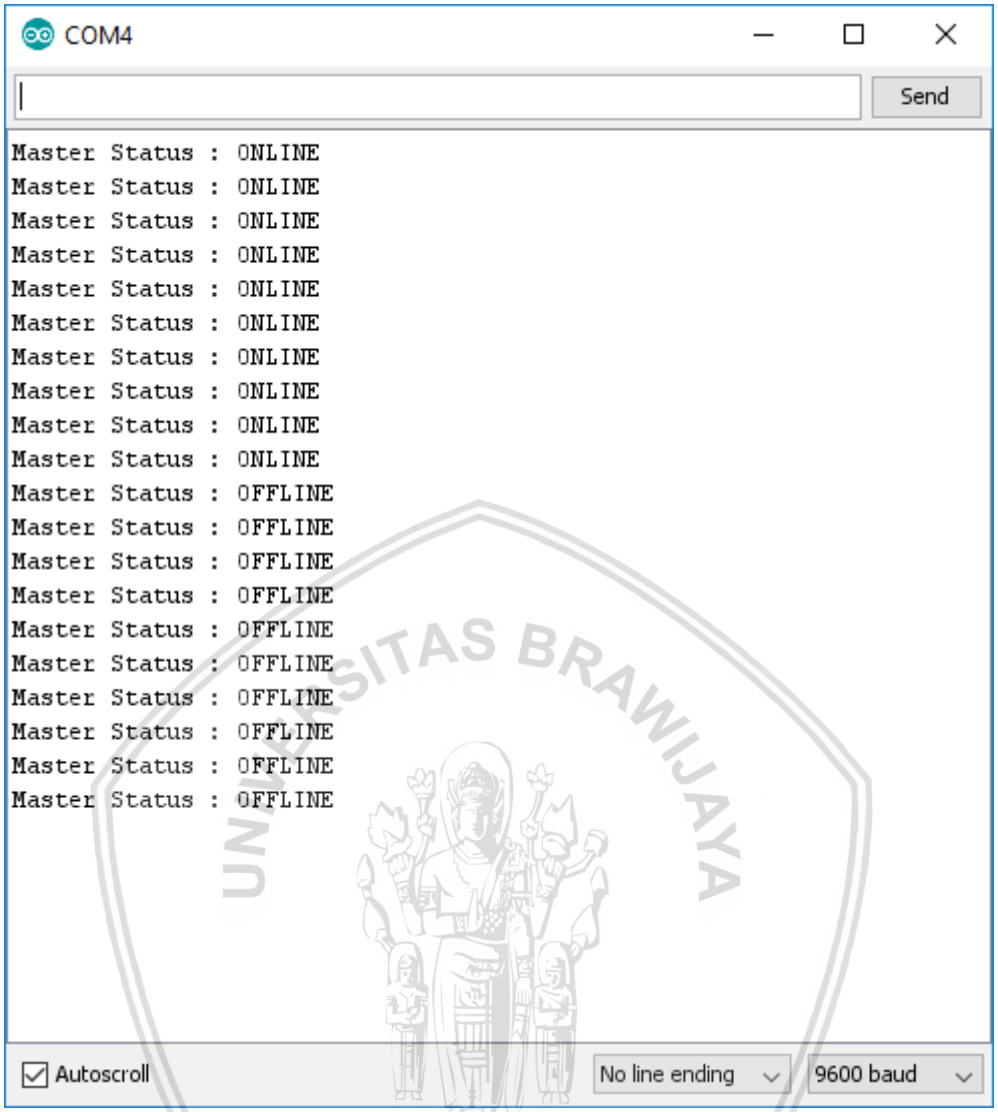
Pengujian Fungsional dilakukan untuk mengetahui apakah kebutuhan fungsional sistem sudah terpenuhi. Sesuai yang tertera pada rekayasa kebutuhan dimana terdapat tiga kebutuhan fungsional sistem yang harus dipenuhi yaitu mengirim *keep-alive status*, membaca *keep-alive status*, dan *switching*. Untuk mengirim dan membaca *keep-alive status* akan dilakukan dalam satu pengujian yaitu pengujiian transaksi *keep alive status.* Hal ini karena tidak memungkinkan menggunakan serial monitor pada *master controller* yang sudah terpasang sistem redundansi. Oleh karena itu pengujian fungsional akan dilakukan dengan dua pengujian yaitu pengujian transaksi *keep-alive status* dan pengujian *switching*.

6.1.1 Pengujian transaksi *keep-alive status*

Pengujian transaksi *keep-alive status* dilakukan untuk melihat apakah pengiriman dan pembacaan keep alive status berjalan dengan benar atau tidak. Pengujian ini dilakukan dengan memantau transaksi melalui serial monitor dari sisi penerima (*fault detector*).

Tahapan pengujian ini cukup hanya dengan menghidupkan sistem kemudian melihat *keep-alive status* yang dikirimkan *master controller* melalui konsol serial monitor Arduino IDE. Sebelum pengujian dilakukan harus dipastikan terlebih dahulu fault detector terhubung dengan Laptop atau PC. Pada saat pengujian berlangsung *master controller* akan dimatikan untuk melihat perubahan *keep-alive status* yang dikirimkan pada *fault detector*.

Setelah tahapan – tahapan pengujian transaksi *keep-alive status* tersebut hasil dapat dilihat pada serial monitor. Hasil pengujian transaksi *keep-alive status* yang dilakukan pada *master* controller dan fault detector ini dapat dilihat pada gambar 6.1.



Gambar 6.1 Hasil pengujian transaksi *Keep-alive status*

Melihat hasil dari pengujian tersebut dapat dilihat bahwa transaksi *keep-alive status* berjalan dengan baik. Jika diperhatikan hasil pengujian pada gambar 6.1 terdapat dua jenis pesan yang diterima fault detector. "*Master* Status : ONLINE" merupakan pesan yang mengindikasikan bahwa *master* controller beroperasi dengan normal. Sedangkan "*Master* Status : OFFLINE" mengindikasikan bahwa *master* controller mati atau error.

6.1.2 Pengujian *switch*ing

Pengujian *switch*ing merupakan pengujian keseluruhan sistem redundansi secara fisik. Pada pengujian ini akan terlihat apakah sistem redundansi mampu melakukan *switching* dan sistem kunci pintu otomatis tetap berjalan normal atau tidak. Pengujian dilakukan dengan menjalankan skenario pengujian yang telah dibuat pada gambar 3.2 sebanyak 10 kali. Skenario pengujian *switch*ing terdiri dari dua skenario yaitu power off (secara hardware) dan infinite loop (secara software).

6.1.2.1 Pengujian switching secara hardware

Berdasarkan skenario pengujian pada Gambar 3.2 (kiri) tahapan prosedur pengujian *switch*ing secara hardware yang dilakukan adalah sebagai berikut :

1. Hidupkan sistem kunci pintu otomatis beserta sistem redundansi hingga beroperasi normal.
2. Matikan perangkat *master* dengan mencabut catu daya yang terpasang pada *master controller*.
3. Tunggu hingga sistem melakukan *fail over* dan perangkat *slave* beroperasi normal.
4. Pasangkan kembali catu daya untuk perangkat *master*
5. Reset perangkat *fault detector* hingga sistem melakukan *fail back* dan perangkat *master* kembali beroperasi normal.

Setelah melakukan prosedur diatas hasil dari pengujian *switch*ing dengan skenario power off dapat dilihat pada Tabel 6.1

Tabel 6.1 Hasil pengujian *switching* dengan *power off*

Pengujian ke-	*Fail over*	*Fail back*	Fungsi Sistem
1	Sukses	Sukses	Normal
2	Sukses	Sukses	Normal
3	Sukses	Sukses	Normal
4	Sukses	Sukses	Normal
5	Sukses	Sukses	Normal
6	Sukses	Sukses	Normal
7	Sukses	Sukses	Normal
8	Sukses	Sukses	Normal
9	Sukses	Sukses	Normal
10	Sukses	Sukses	Normal

6.1.2.2 Pengujian switching secara software

Berdasarkan skenario pengujian pada Gambar 3.2 (kanan) tahapan prosedur pengujian *switch*ing secara software yang dilakukan sebagai berikut :

1. Hidupkan sistem kunci pintu otomatis beserta sistem redundansi hingga beroperasi normal.
2. Scan key chain yang berfungsi menjebak program kepada infinite loop.

3. Tunggu 5 detik hingga sistem melakukan *fail over* dan perangkat *slave* beroperasi normal.
4. Reset perangkat *fault detector* hingga sistem melakukan *fail back* dan perangkat *master* kembali beroperasi normal.

Setelah melakukan prosedur diatas hasil dari pengujian dengan skenario infinite loop dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil pengujian *switching* dengan *infinite loop*

Pengujian ke-	*Fail over*	*Fail back*	Fungsi Sistem
1	Sukses	Sukses	Normal
2	Sukses	Sukses	Normal
3	Sukses	Sukses	Normal
4	Sukses	Sukses	Normal
5	Sukses	Sukses	Normal
6	Sukses	Sukses	Normal
7	Sukses	Sukses	Normal
8	Sukses	Sukses	Normal
9	Sukses	Sukses	Normal
10	Sukses	Sukses	Normal

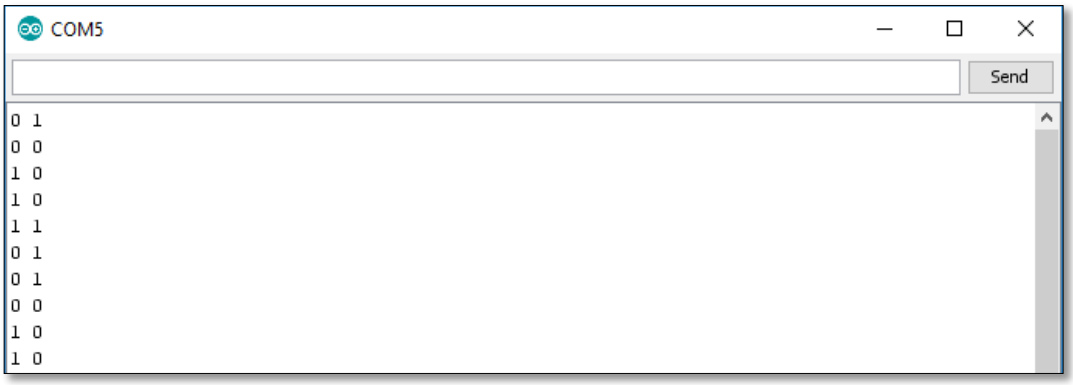
6.2 Pengujian Non Fungsional

Pengujian non fungsional dilakukan untuk menguji kebutuhan sistem diluar dari fungsi tujuan sistem. Dalam hal ini akurasi diuji dari pengiriman serta pembacaan *Keep-alive status* antara perangkat *master* dengan *fault detector*. Selain akurasi pengujian juga dilakukan pada jeda waktu error dengan *switch*ing *fail over* dan jeda waktu hingga sistem siap dioperasikan.

6.2.1 Pengujian akurasi *Keep-alive status*

Pengujian akurasi *Keep-alive status* dilakukan dengan menampilkan nilai dua variabel yang menyimpan nilai *Keep-alive status* yg selalu berubah – ubah. Nilai akurasi ditentukan dengan melihat nilai kedua variabel tersebut. Jika nilainya berbeda maka artinya akurat, jika sama maka tidak akurat.

Pengujian dilakukan dengan melihat penerimaan pada sisi *fault detector* melalui serial monitor Arduino IDE selama 5 detik. Dari 5 detik tersebut terjadi 10 transaksi *Keep-alive status* yang terjadi. Hasil pengujian dapat dilihat pada Gambar 6.2.



Gambar 6.2 Hasil pengujian akurasi transaksi *Keep-alive status*

Hasil tersebut menunjukkan 7 dari 10 transaksi yang akurat. Angka tersebut menandakan bahwa dari 5 detik operasi sistem redundansi terdapat 1,5 detik (3 kali 0,5 detik tidak berturut – turut) transaksi yang tidak akurat. Namun dengan tingkat akurasi tersebut sudah membuat sistem redundansi tidak mengalami kegagalan *switch*ing sama sekali dalam 20 kali percobaan (secara hardware dan software).

6.2.2 Pengujian jeda *switch*ing

Pengujian jeda *switch*ing dilakukan untuk mengetahui berapa lama jeda *switch*ing dan berapa lama jeda yang dibutuhkan setelah *switch*ing sampai sistem dapat berfungsi normal. Jeda yang diberikan pada program adalah 5 detik untuk mempertimbangkan keputusan *switch*ing.

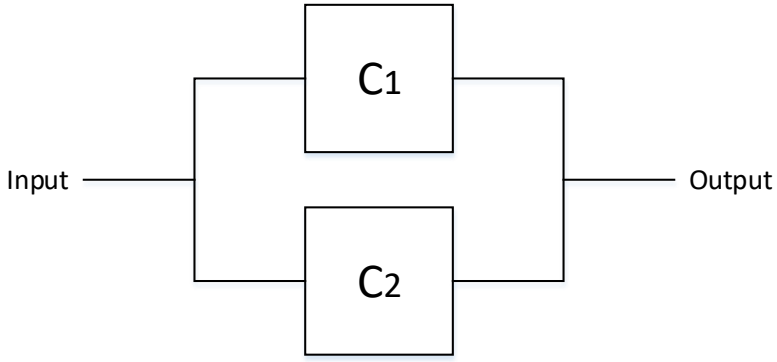
Pengujian dilakukan dengan bantuan fitur stopwatch pada ponsel genggam untuk menghitung waktu sebenarnya. Setelah dilakukan pengujian waktu stopwatch menunjukkan 1 detik lebih lambat dari waktu jeda yang disematkan pada program yaitu 6 detik. Sedangkan untuk jeda waktu setelah *switch*ing sampai sistem siap digunakan adalah 2 detik. Dengan demikian waktu jeda total dari saat error terjadi hingga perangkat *slave* siap beroperasi adalah 8 detik.

6.3 Analisis Reliability

Hasil pengujian yang sudah dilakukan akan dianalisis untuk membuktikan apakah *reliability* sistem dapat meningkat setelah ditambahkan sistem redundansi. Analisis dilakukan dengan menggunakan pemodelan *dependability* sistem. Dengan pemodelan *dependability* akan dapat diketahui peningkatan *reliability* pada sistem.

6.3.1 *Reliability Block Diagram* (RBD)

Berdasarkan desain sistem yang telah dibuat dan meninjau kembali tinjauan pustaka tentang evaluasi dependability didapatkan RBD yang sesuai untuk mengevaluasi *reliability* sistem. *Reliability* block diagram dapat dilihat pada Gambar 6.2.



Gambar 6.3 Hasil pemodelan *Reliability Block Diagram* sistem

6.3.2 Evaluasi *Reliability*

Berdasarkan nilai failure rate Arduino Uno hasil prediksi yang dilakukan oleh George Novacek dengan metode Bellcore V6 yaitu 2.0559 x 10^{-5} per jam, nilai tersebut dapat dimasukkan pada persamaan 2.2 untuk mengetahui *reliabilitiy*. Jika diasumsikan bahwa nilai tersebut merupakan *failure rate* Arduino Uno maka dalam misi operasi 1 tahun sesuai garansi dari Arduino didapatkan perhitungan nilai *reliability* Arduino Uno sebagai berikut.

$$R(t) = e^{-\lambda t}$$

$$R(t) = e^{-2.0559\times10^{-5}\times8760}$$

$$R(t) = 0.8351$$

Dari perhitungan tersebut dapat diketahui bahwa nilai *reliability* Arduino uno dalam masa operasi 1 tahun adalah 83.51%.

Berdasarkan RBD yang sudah dimodelkan pada Gambar 6.3 dapat dilihat bahwa sistem ini merupakan sistem parallel. Dalam sistem yang parallel dibutuhkan setidaknya satu komponen bekerja. Berdasarkan persamaan sistem paralel yang terdapat pada persamaan 2.4 perhitungan *reliability* yang didapatkan untuk mengevaluasi *reliability* sebagai berikut:

$$R(t) = 1 - \prod_{i=1}^{n}(1 - Ci(t))$$

$$R(t) = 1 - \big(1 - C_1(t)\big)\big(1 - C_2(t)\big)$$

$$R(t) = 1 - (1 - 0.8351)(1 - 0.8351)$$

$$R(t) = 1 - (0.1024 \times 0.1024)$$

$$R(t) = 1 - 0.0104$$

$$R(t) = 0.9896$$

Dari hasil perhitungan diatas didapatkan nilai R(t) = 0.9896 atau dengan kata lain sistem kunci pintu otomatis berbasis Arduino Uno dengan *Cold Standby Redundancy* memiliki nilai *reliability* 15,45%.

Apabila sistem kunci pintu otomatis tanpa *Cold Standby Redundancy* maka akan menjadi sistem seri. Berdasarkan persamaan sistem seri yang terdapat pada persamaan 2.3 perhitungan *reliability* yang didapatkan untuk mengevaluasi *reliability* sebagai berikut:

$$R(t) = \prod_{i=1}^{n} Ci(t)$$

$$R(t) = Ci(t)$$

$$R(t) = 0.8351$$

Dari perhitungan tersebut menghasilkan nilai R(t) = 0.8351 atau dengan kata lain sistem kunci pintu otomatis berbasis Arduino Uno tanpa redundansi memiliki nilai *reliability* 83.51%.

Setelah melihat perbandingan hasil perhitungan *reliability* sistem kunci pintu otomatis tanpa redundansi dengan yang menggunakan *Cold Standby Redundancy* dapat dilihat bahwa sistem dengan redundansi memiliki nilai *reliability* yang lebih tinggi. Setelah menambahkan sistem redundansi nilai *reliability* sistem meningkat sebesar 15,45% dalam 1 tahun beroperasi.

BAB 7 PENUTUP

Pada bab ini akan dijelaskan kesimpulan terhadap hasil dari penelitian yang telah dilakukan. Kesimpulan akan dijelaskan berdasarkan rumusan masalah dengan hasil penelitian. Saran – saran juga akan diberikan pada bab ini untuk tujuan penelitian lebih lanjut.

7.1 Kesimpulan

Melihat kembali rumusan masalah pada awal bab dan hasil penelitian pada analisis pengujian serta perancangan sistem penulis dapat menarik beberapa poin kesimpulan. Berikut kesimpulan yang didapat dari hasil penelitian :

1. Merancang perangkat keras sistem hardware redundancy dengan metode *Cold Standby* membutuhkan 3 komponen utama yaitu redundant unit (*slave*), fault detector, dan juga *switch*. Ketiga komponen tersebut harus dikonfigurasi sesuai fungsi masing masing. Redundant unit merupakan duplikat dari komponen utama (*master*) yang diredundansi sehingga konfigurasinya cukup dengan menyamakan konfigurasi pin I/O milik *master*. Fault detector memiliki peran sangat penting sehingga harus memiliki tangkat reliabilitas yang sangat tinggi karena harus memantau dan memutuskan untuk *switch*ing ketika sistem *fail.* Konfigurasi hardware untuk fault detector yang terutama adalah hubungan komunikasi Serial I2C melalui pin SDA dan SCL karena berhubungan dengan *keep-alive status* dari *master* controller. *Switch* (relay) yang bertugas merubah jalur digital I/O dari *master* ke *slave* dan sebaliknya juga harus dikonfigurasi dengan teliti karena terdapat banyak sekali *wiring* yang rawan terjadi kesalahan. Konfigurasi *switch* adalah dengan memasangkan digital I/O *master* pada kanal NC relay, digital I/O *slave* pada kanal NO relay dan I/O device pada kanal COM relay.
2. Rancangan perangkat lunak dilakukan pada redundant unit dan fault detector. Seperti halnya konfigurasi hardware redundant unit dalam konfigurasi software juga hanya perlu menduplikat program dari *master*. Fault detector berkomunikasi melalui antarmuka serial I2C dengan *master* untuk transaksi *keep-alive status*. Untuk melakukan transaksi tersebut keduanya membutuhkan *library* wire dan di sisi *master* ditambahkan *library VirtualDelay* agar pengiriman *keep-alive status* berjalan secara multitask. Berdasarkan hasil pengujian akurasi dan timeout *switch*ing ternyata terjadi perbedaan jeda dimana konfigurasi pada program jeda hanya 5 detik namun *switch*ing baru terjadi pada detik ke 6. Hal ini mungkin disebabkan oleh tingkat akurasi transaksi *keep-alive* yang masih belum maksimal yaitu 70%. Namun hasil tersebut tidak mengganggu sistem redundansi yang berjalan lancar selama 20 kali pengujian (secara hardware dan software) tanpa ada kegagalan satu pun.
3. Melihat kembali hasil analisis perhitungan *reliability* sistem tanpa redundansi dengan sistem yang menggunakan redundansi dapat disimpulkan bahwa sistem dengan redundansi memiliki nilai *reliability* yang lebih tinggi. Hal ini

membuktikan bahwa sistem paralel lebih reliabel daripada sistem seri. Hasil analisis menunjukkan peningkatan *reliability* sebesar 15.45% dengan nilai *failure rate* Arduino Uno 2.0559 x 10^{-5} per jam.

7.2 Saran

Setelah menyimpulkan beberapa hal mengenai perancangan hardware dan software sistem penulis ingin memberikan beberapa saran untuk penelitian lebih lanjut. Beberapa saran yang dapat penulis berikan yaitu:

1. *Switch* device yang digunakan dalam penelitian ini terlalu mahal dan besar, untuk pengembangan selanjutnya disarankan untuk memilih *switch* device yang lebih sederhana dan dengan harga yang lebih terjangkau.
2. Untuk penelitian lebih lanjut disarankan menggunakan *fault detector* dengan tingkat *reliability* yang lebih tinggi karena perannya yang sangat krusial dalam *Cold Standby Redundancy*.
3. Untuk pengembangan lebih lanjut mungkin bisa ditambahkan unit *spare* untuk membandingkan bagaimana analisis *reliability* sistem yang menggunakan *Cold Standby Redundancy* dengan 2 perangkat *spare*.

DAFTAR PUSTAKA

Albert, v. D., 2017. *Non-Blocking Virtual Delay Timer for the Arduino.* [Online] Tersedia di at: http://www.avdweb.nl/arduino/libraries/virtualdelay.html [Diakses pada 25 November 2017].

Arduino.cc, 2017. *Arduino : ARDUINO NANO.* [Online] Tersedia di at: https://store.arduino.cc/arduino-nano [Diakses pada 14 November 2017].

Arduino.cc, 2017. *Arduino : ARDUINO UNO REV3.* [Online] Tersedia di at: https://store.arduino.cc/arduino-uno-rev3 [Diakses pada 13 November 2017].

Arduino.cc, 2017. *Arduino : Introduction.* [Online] Tersedia di at: https://www.arduino.cc/ [Diakses pada 8 September 2017].

Arduino.cc, 2017. *Arduino : Wire Library.* [Online] Tersedia di at: https://www.arduino.cc/en/reference/wire [Diakses pada 25 November 2017].

Arduino.cc, 2017. *Arduino/Genuino Products Warranty.* [Online] Tersedia di at: https://www.arduino.cc/en/Main/warranty [Diakses pada 10 Desember 2017].

Arduino.cc, 2017. *Arduino: Environment.* [Online] Tersedia di at: https://www.arduino.cc/en/guide/environment [Diakses pada 8 September 2017].

Dubrova, E., 2013. *Fault-Tolerant Design.* New York: Springer Science+Business Media.

Johnson, B., 1984. Fault-Tolerant Microprocessor-Based Systems. *IEEE Micro,* Volume 4, pp. 6-21.

Lapric, J., 1985. Dependable computing and fault tolerance: Concepts and terminology In: Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15). *IEEE Computer Society,* pp. 2-11.

Mathew, M. & Divya, R. S., 2017. Survey on Various Door Lock Access Control Mechanisms. *International Conference on circuits Power and Computing Technologies [ICCPCT] ,* pp. 1-3.

NASA, 2000. The Role of Small Satellites in NASA and NOAA Earth Observation Programs. *.Space Studies Board, National Research Council, National Academy of Sciences, Washington, USA .*

National Instruments, 2008. *Redundant System Basic Concepts.* [Online] Tersedia di at: http://www.ni.com/white-paper/6874/en/ [Diakses pada 8 September 2017].

Nedelkovski, D., 2017. *How RFID Works and How To Make an Arduino based RFID Door Lock.* [Online] Tersedia di at: https://howtomechatronics.com/tutorials/arduino/rfid-works-make-arduino-based-rfid-door-lock/ [Diakses pada 5 November 2017].

Nehete, P. R., Chaudhari, J., Pachpande, S. & Rane, K., 2016. Literature Survey on Door Lock Security Systems. *International Journal of Computer Applications,* Volume 153-No 2, pp. 13-18.

Novacek, G., 2016. *Tips for Predicting Product Reliability.* [Online] Tersedia di at: http://circuitcellar.com/cc-blog/tips-for-predicting-product-reliability/ [Diakses pada 9 Juli 2018].

Roosano, A. A. & Purnomo, J., 2016. Desain dan Prototipe Kunci Pintu Otomatis Menggunakan RFID Berbasis Arduino Uno. *Jurnal Ilmiah Informatika dan Komputer,* Volume 21, p. 8.

Saputro, E. & Wibawanto, H., 2016. Rancang Bangun Pengaman Pintu Otomatis Menggunakan E-KTP Berbasis Mikrokontroler ATmega328. *Jurnal Teknik Elektro,* Volume 8 No. 1, pp. 1-4.

Siewiorek, D. & Swarz, R., 1998. *Reliable Computer Systems Design and Evaluation.* 3rd ed. Wellesley : A K Peters Ltd..

Weiser, M., 1993. Somecomputerscienceproblemsinubiquitouscomputing. *Commun.ACM,* Volume 36, pp. 74-83.